ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ

ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΜΠΑΓΗΣ ΑΝΑΠΑΡΑΣΤΑΣΗ ΚΑΙ ΕΠΕΞΕΡΓΑΣΙΑ ΓΡΑΦΗΜΑΤΩΝ ΜΕΓΑΛΟΥ ΜΕΓΕΘΟΥΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

ΤΣΙΑΝΑΚΑ ΕΥΘΑΛΙΑ

ΕΠΙΒΛΕΠΩΝ:

ΒΑΣΙΛΑΚΟΠΟΥΛΟΣ ΜΙΧΑΗΛ, ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ

Βόλος, 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΜΠΑΓΗΣ ΑΝΑΠΑΡΑΣΤΑΣΗ ΚΑΙ ΕΠΕΞΕΡΓΑΣΙΑ ΓΡΑΦΗΜΑΤΩΝ ΜΕΓΑΛΟΥ ΜΕΓΕΘΟΥΣ

COMPACT REPRESENTATION OF WEB GRAPHS WITH EXTENDED FUNCTIONALITY

Διπλωματική εργασία που υποβλήθηκε στην Πολυτεχνική Σχολή του Πανεπιστημίου ΘΕΣΣΑΛΙΑΣ σαν μέρος των υποχρεώσεων για την απόκτηση του μεταπτυχιακού διπλώματος <<Επιστήμη και τεχνολογία ΗΜΜΥ>> .

Τσιανάκα Ευθαλία

Επιβλέπων:
Βασιλακόπουλος Μιχαήλ
Αναπληρωτής Καθηγητής Πανεπιστημίου Θεσσαλίας

Εγκρίθηκε από την τριμελή επιτροπή

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
…….	…….	…….
Βασιλακόπουλος Μιχαήλ Αναπλ. Καθηγητής Π.Θ.	Μποζάνης Παναγιώτης Καθηγητής Π.Θ.	Τσομπανοπούλου Παναγιώτα Αναπλ. Καθηγήτρια

ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτα απ’ όλα, θέλω να ευχαριστήσω τον επιβλέποντα της μεταπτυχιακής εργασίας μου, Αναπληρωτή Καθηγητή κ. Βασιλακόπουλο Μιχαήλ, για την πολύτιμη βοήθεια και καθοδήγησή του κατά τη διάρκεια της δουλειάς μου, καθώς και τα υπόλοιπα μέλη της επιτροπής.

Επιπλέον, θα ήθελα να ευχαριστήσω την μητέρα μου Σωτηρία Τσιανάκα και τα αδέρφια μου Γεωργία Τσιανάκα και Βασίλειο Τσιανάκα για την υποστήριξή τους όλα αυτά τα χρόνια αλλά και για τις αξίες που μου μεταβίβασαν .

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια, η ανάπτυξη της τεχνολογίας είναι ραγδαία κι επακόλουθο αυτού είναι η αύξηση του όγκου των δεδομένων που πρέπει να χειριστούμε και να επεξεργαστούμε (π.χ. ένας τομέας που συναντάμε μεγάλο όγκο δεδομένων είναι αυτός του διαδικτύου, πιο συγκεκριμένα η αναπαράσταση μεγάλων υποσυνόλων του παγκόσμιου ιστού, World Wide Web). Για αυτόν τον λόγο έχει δημιουργηθεί η ανάγκη για τη δημιουργία μεθόδων γρήγορων αναζητήσεων σε τέτοια δεδομένα.

Ένας τρόπος αναπαράστασης δεδομένων (π.χ. του παγκόσμιου ιστού ή των κοινωνικών δικτύων) είναι τα γραφήματα και συνήθως ο όγκος των δεδομένων που περιέχουν γραφήματα από συστήματα όπως τα προηγούμενα είναι πολύ μεγάλος, με αποτέλεσμα οι αναπαραστάσεις τους να μη «χωρούν» σε έναν συνηθισμένο υπολογιστή. Για αυτό τον λόγο έχουν αναπτυχθεί διάφοροι τρόποι συμπίεσης των γραφημάτων. Η συμπιεσμένη αναπαράσταση των γραφημάτων έχει μειώσει αρκετά τον χώρο που καταλαμβάνουν, παρόλο που έχει επίδραση στο χρόνο αναζήτησης δεδομένων σε αυτά. Μια μέθοδος που συνδυάζει τη συμπίεση, χωρίς να έχει μεγάλη επίπτωση στο χρόνο αναζήτησης είναι το k^2 – δένδρο.

Σε αυτή τη διπλωματική εργασία περιγράφεται και υλοποιείται η δημιουργία ενός k^2 – δένδρου από έναν πίνακα γειτνιάσεως, ο οποίος αποτελείται από μηδενικά και άσσους (ύπαρξη και απουσία ακμής μεταξύ κόμβων, αντίστοιχα). Επιπλέον, παρουσιάζονται και υλοποιούνται μέθοδοι περαιτέρω συμπίεσης μεγάλων γραφημάτων στην κύρια μνήμη, επεκτάσεις του k^2 – δένδρου, ώστε να χωρούν στη μνήμη ακόμη μεγαλύτερα γραφήματα και ταυτόχρονα να γίνονται αρκετά γρήγορα οι αναζητήσεις για διάφορα ερωτήματα (Queries). Επιπροσθέτως, εκτελούνται συγκεκριμένα ερωτήματα για να ελεγχθεί, κυρίως σε χρόνο αλλά και σε χώρο, η αποτελεσματικότητα της συμπίεσης που πραγματοποιήθηκε. Τέλος, πραγματοποιούνται κάποια πειράματα για να ελέγξουμε αν η συμπίεση που έχει επιτευχθεί είναι αρκετά αποτελεσματική όσον αφορά τον συνολικό χρόνο «τρεξίματος» των διάφορων ερωτημάτων.

Λέξεις κλειδιά: K^2 -δένδρα, συμπιεσμένα γραφήματα, πίνακας γειτνιάσεως, μέθοδοι συμπίεσης γραφημάτων, πίνακας T, πίνακας L,ερωτήματα.

ABSTRACT

During the last years the technology has been developed rapidly. As a consequence, there has been on increase on the amount of data which is to be processed (for instance the Internet is a place where one can find a huge amount of data and more specifically the representation of large subsets of the World Wide Web). For this reason there has been a need to create quick searches methods for such data.

A way to represent data (for example data related with world wide web or social networks) is using graphs and usually the amount of these data, which contain graphs from systems as above, is so big that their representations do not "fit" in a typical computer. Therefore, there have been developed several methods to compress these graphs in main memory. The compressed graph representation has quite reduced the space they take despite the fact that has an impact on search time. A method that combines the compression without having a major impact on search time is the k^2-tree.

The target of this master's thesis is to describe and implement the structure of a k^2- tree from an adjacency matrix which consists of zeroes and aces (existence and absence of edge between nodes, respectively). Furthermore, methods of further compression of large graphs in main memory (k^2-tree extensions) are presented and implemented so that even larger graphs can be stored in main memory and, at the same time, the searches on various queries we choose to be quick enough. Additionally, we execute specific queries to check the compression efficiency regarding time and space. Finally, the effect of this compression on execution time of several queries is checked through experiments.

Keywords: K^2-trees, compressed graphs, adjacency matrix, compression methods, matrix T, matrix L, queries.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

1. ΕΙΣΑΓΩΓΗ

Οι συμπιεσμένες γραφικές αναπαραστάσεις (compact graph representation) είναι δομές δεδομένων οι οποίες αναπαριστούν μεγάλου μεγέθους γραφήματα κι υποστηρίζουν γρήγορες αναζητήσεις χωρίς να χρειάζεται να αποσυμπιεστούν. Στόχος τους είναι να υποστηρίζουν πολύπλοκους αλγορίθμους σε μεγάλου μεγέθους γραφήματα χρησιμοποιώντας τη βασική μνήμη.

Σε αυτή την εργασία εισάγουμε μια νέα έννοια αυτή του k^2-δένδρου, που είναι μιας συμπαγής αναπαράσταση του πίνακα γειτνιάσεως, η οποία υποστηρίζει προς τα εμπρός αλλά και προς τα πίσω αναζητήσεις. Επιπλέον, υποστηρίζει κι άλλα πιο σύνθετα ερωτήματα, όπως για παράδειγμα ο έλεγχος για το αν δύο κόμβοι είναι γειτονικοί ή ακόμη και αν υπάρχει κάποιος σύνδεσμος μεταξύ ενός εύρους κόμβων.

Η μέθοδος του k^2-δένδρου εκμεταλλεύεται το γεγονός πως μεγάλα υποσύνολα του πίνακα γειτνιάσεως αποτελούνται από μηδενικά ή μονάδες. Γενικά υπάρχουν πολλοί μέθοδοι συμπίεσης κι ενώ μπορεί να είναι πιο γρήγορες σε σχέση με αυτή που πραγματευόμαστε εμείς, καταλαμβάνουν πολύ χώρο στην κύρια μνήμη. Η μέθοδος μας έχει αρκετά ικανοποιητικά αποτελέσματα από άποψη χώρου αλλά και χρόνου.

1.1 Αναφορά σε διάφορες λειτουργίες των Γραφημάτων Ιστού (Web graphs)

Στα γραφήματα ιστού οι κόμβοι αναπαριστούν κάθε ιστοσελίδα ([4]) κι οι ακμές τους υπερσυνδέσμους ([5]), αλλά είναι πιθανό οι κόμβοι να αναπαριστούν domains ή servers [6]. Σε αυτή την παράγραφο θα αναφερθούμε σε διάφορες αναλύσεις των γραφημάτων ιστού. Για παράδειγμα το άρθρο Mining the inner structure of the Webgraph [7] μας δείχνει πως συνηθισμένες τεχνικές χρησιμοποιούνται για την ανακάλυψη της δομής των γραφημάτων ιστού, οι οποίες βασίζονται σε κλασικούς αλγόριθμους όπως οι depth-first search [8] και breath-first search [9]. Μια πολύ σημαντική ανάλυση που μπορεί να γίνει μέσω των γραφημάτων ιστού είναι η μέτρηση της σημασίας των

ιστοσελίδων. Για παράδειγμα ένας από τους πιο σημαντικούς αλγόριθμους για την εύρεση των κόμβων (hubs) και των αρχών (authorities) στον ιστό (web), HITS [10], ξεκινά επιλέγοντας τυχαία σελίδες (pages) και βρίσκοντας τα επαγόμενα δευτερεύοντα γραφήματα, τα οποία είναι σελίδες που υποδεικνύουν ή υποδεικνύονται από τις επιλεγμένες σελίδες.

Επιπλέον, με την ανάλυση των γραφημάτων ιστού μπορεί να αντιμετωπιστεί ένα ακόμη σημαντικό πρόβλημα, αυτό της εύρεσης του Web spam [11]. Το άρθρο Link analysis for Web spam detection [12] αναφέρεται σε χρήσιμες τεχνικές για την ανίχνευση του Web Spam, οι οποίες απαιτούν ικανότητες για προς τα εμπρός αλλά και προς τα πίσω αναζήτηση. Γενικά τα γραφήματα διευκολύνουν τη διαχείριση του μεγάλου όγκου δεδομένων που μπορεί να συναντήσουμε σε διάφορους τομείς.

1.2 Παρουσίαση διάφορων μεθόδων συμπίεσης γραφημάτων

Οι περισσότερες έρευνες, για τη συμπίεση των γραφημάτων ιστού, επικεντρώνονται στη διατήρηση της συμπαγούς αναπαράστασης, ώστε να είναι δυνατή η εξαγωγή των successors (δηλαδή αν υπάρχει σύνδεσμος μεταξύ δύο κόμβων) κάθε ιστοσελίδας (Web page). Ο χώρος που απαιτείται για αυτές τις μεθόδους μετριέται σε bits ανά ακμή (bits per edge, bpe), δηλαδή το πλήθος των bits που είναι απαραίτητα για να λειτουργεί το γράφημά μας στην κύρια μνήμη διαιρούμενο από το πλήθος των ακμών του γραφήματος.

Πάνω σε αυτό τον τομέα μια πολύ σημαντική δουλειά αναφέρεται από τους Boldi και Vigna στο άρθρο The WebGraph framework I: compression techniques [13]. Η μέθοδός τους ονομάζεται WebGraph compression και σύμφωνα με τις στατιστικές τους αναλύσεις είναι μια πολύ επιτυχημένη μέθοδος. Επιτρέπει την γρήγορη εξαγωγή των γειτόνων ενός κόμβου (page), σε περίπου 1-6 bpe. Η αναπαράστασή τους «εκμεταλλεύεται» πολλά χαρακτηριστικά των γραφημάτων όπως για παράδειγμα το «αντίγραφο ιδιοκτησίας» (copy property), δηλαδή το σύνολο των γειτόνων μιας σελίδας που είναι όμοιο με αυτό ενός άλλου κόμβου.

Οι Asano, Miyawaki και Nishizeki [14] συμπίεσαν τις ήδη γνωστές ιδιότητες (όπως γραμμές, στήλες, διαγωνίους, κ.α.) ενός πίνακα γειτνιάσεως. Με αυτόν τον τρόπο κατάφεραν να έχουν καλά αποτελέσματα, σε μικρά γραφήματα, όσον αφορά τον χρόνο συμπίεσης, αλλά δεν είχαν εξίσου καλά αποτελέσματα στον χρόνο αναζήτησης (navigation). Συνήθως οι κόμβοι ενός δένδρου διατάσσονται με αλφαβητική σειρά (αν περιέχουν λέξεις ή γράμματα). Οι Apostolico και Drovandi [15] παρουσίασαν μια μέθοδο, η οποία διατάσσει τους κόμβους ενός γραφήματος σύμφωνα με τη μέθοδο

Breadth First Search [16]. Με αυτόν τον τρόπο καταφέρνουν να δεσμεύουν λιγότερο χώρο ακόμη κι από αυτόν που παρουσιάστηκε στο άρθρο [13], διατηρώντας παράλληλα κι έναν καλό χρόνο ανάκτησης. Επιπροσθέτως, εισάγουν μια αποτελεσματική τεχνική για την εύρεση συνδέσμων μεταξύ των κόμβων.

Στο άρθρο τους οι Buehrer και Chellapilla προτείνουν μια μέθοδο συμπίεσης των γραφημάτων ιστού που βασίζεται στην εύρεση των bicliques [18] σε έναν κατευθυνόμενο γράφο. Ονόμασαν τη μέθοδό τους «Virtual Node Miner (VMN)» κι είναι χτισμένη έτσι ώστε να επιδέχεται αυξανόμενες ενημερώσεις. Η VMN θα μπορούσε να περιγραφεί ως μια μέθοδος προεπεξεργασίας γραφημάτων, που μετά την υλοποίηση της ξαναπαίρνουμε ένα βελτιωμένο γράφημα πάνω στον οποίο μπορούν εφαρμοστούν διάφοροι μέθοδοι συμπίεσης, όπως αυτοί που θα περιγράψουμε αργότερα.

Σαν πρότυπα για τη δημιουργία της συγκεκριμένης εργασίας υπήρξαν τα άρθρα Compact representation of Web graphs with extended functionality [1] και Compact Querieable Representations of Raster Data [2]. Στο άρθρο [1] οι συγγραφείς αναφέρουν πως χρησιμοποιούν τη συμπιεσμένη μορφή γράφων για την αναπαράσταση των γραφημάτων ιστού (Web Graphs [3]), τα οποία είναι πάρα πολύ μεγάλα σε όγκο. Σύμφωνα με κάποια πειράματα που έχουν πραγματοποιήσει ισχυρίζονται πως το k^2-δένδρο είναι η πιο συμπιεσμένη αναπαράσταση γραφημάτων ιστού αφού η αναζήτηση προς τα εμπρός αλλά και προς τα πίσω φτάνει τα 1-3 bits ανά σύνδεσμο (link). Βέβαια υπάρχουν και πιο γρήγοροι μέθοδοι αλλά απαιτούν πάρα πολύ χώρο σε σχέση με τη δική τους αναπαράσταση. Στο άρθρο [2] περιγράφεται με λεπτομέρεια η μέθοδος συμπίεσης στις μονάδες, ενώ το άλλο άρθρο [1] αναφέρεται στη μέθοδο συμπίεσης των μηδενικών.

Είναι απαραίτητο να αναφερθεί πως σε αυτή την εργασία δε μας ενδιαφέρει ο χρόνος που χρειάζεται να φτιαχτεί ο πίνακας γειτνιάσεως αλλά επικεντρωνόμαστε στο χρόνο που χρειάζεται για να φτιαχτούν οι πίνακες T και L (θα αναφερθούμε εκτενέστερα σε αυτούς παρακάτω), δηλαδή οι πίνακες που χρησιμοποιούμε για την αναπαράσταση της συμπιεσμένης μορφής γραφημάτων που προτείνουμε. Και βεβαίως μας ενδιαφέρει ο χρόνος εκτέλεσης διάφορων ερωτημάτων που θέτουμε.

Η μέθοδος που ακολουθήθηκε σε αυτήν την εργασία προτείνει τη συμπίεση των μονάδων και μηδενικών, το οποίο θα μας αποφέρει καλύτερους χρόνους εκτέλεσης αλλά και εξοικονόμηση χώρου. Αυτή η μέθοδος είναι αποτελεσματική αφού ένας ολόκληρος υποπίνακας γεμάτος με μηδενικά ή άσσους απεικονίζεται με μόνο ένα ψηφίο (0 ή 1), με αποτέλεσμα ο χρόνος αναζήτησης να μειώνεται πολύ (δεν κάνουμε αναζήτηση σε έναν ολόκληρο πίνακα). Για τους ίδιους λόγους μειώνεται κι ο χώρος που καταλαμβάνεται στην κύρια μνήμη, αφού ολόκληρος ο πίνακας γειτνιάσεως αναπαρίσταται σε μόνο δύο πίνακες, τους συμπιεσμένους T και L. Το άρθρο [1] που είναι η πηγή αυτής της εργασίας

επικεντρώνεται στα γραφήματα ιστού (αφού εκεί συναντάμε δισεκατομμύρια) και πως θα γίνει συμπίεση αυτών, το ίδιο και τα άρθρα που αναφέρθηκαν παραπάνω. Βέβαια αυτή η μέθοδος μπορεί να χρησιμοποιηθεί για διάφορα ενδεχόμενα (θα παρουσιαστούν κάποια παρακάτω).

1.3 Δομή της εργασίας

Σε αυτή την εργασία εισάγεται η έννοια του k^2-δένδρου (k^2-tree), η οποία είναι μια συμπιεσμένη δενδρική αναπαράσταση (compact tree representation) ενός πίνακα γειτνιάσεως που υποστηρίζει πλοήγηση προς τα εμπρός αλλά και προς τα πίσω. Επιπλέον, υποστηρίζει πιο εξελιγμένες λειτουργίες, όπως η εύρεση συνδέσμου, δηλαδή αν υπάρχει ακμή, μεταξύ κόμβων, απαρίθμηση των συνδέσμων σε ένα εύρος κόμβων και πολλές άλλες λειτουργίες στις οποίες θα αναφερθούμε εκτενέστερα παρακάτω.

Το κύριο μέρος αυτής της εργασίας αποτελεί η ανάπτυξη και κωδικοποίηση μεθόδων για τη δημιουργία ενός συμπιεσμένου γράφου, αλλά και την εκτέλεση διάφορων ερωτημάτων για να ελεγχθεί η αποτελεσματικότητα της συμπίεσης που εκτελέσαμε. Οι μέθοδοι συμπίεσης που ακολουθούμε πετυχαίνουν πολύ καλή αναλογία συμπίεσης κι αποτελεσματικής πλοήγησης, εξαιτίας της απλότητας των δομών που χρησιμοποιούμε για την αναπαράσταση του k- δένδρου. Ο κώδικας για την υλοποίηση της συμπίεσης γράφτηκε σε ένα ολοκληρωμένο περιβάλλον προγραμματισμού το Code Blocks [19], το οποίο παρέχεται σε αυτήν την ιστοσελίδα http://www.codeblocks.org/downloads . Τέλος, η γλώσσα που χρησιμοποιήθηκε είναι η C [20].

Στο 2^o κεφάλαιο αυτής τη εργασίας θα εξηγηθεί ο τρόπος δημιουργίας του k^2- δένδρου από τον πίνακα γειτνιάσεως, αλλά και των πινάκων T και L. Δηλαδή θα περιγραφούν λεπτομερώς οι μέθοδοι συμπίεσης που ακολουθήσαμε. Επίσης, θα παρουσιαστεί και θα αναλυθεί ο κώδικας που γράφτηκε για την υλοποίηση αυτών των μεθόδων αλλά και το χτίσιμο των πινάκων T και L. Το Κεφάλαιο 3 περιέχει την υλοποίηση διάφορων ερωτημάτων (queries) για να ελεγχθεί η αποδοτικότητα των συμπιέσεων που πραγματοποιήσαμε. Επίσης, για να ελεγχθεί η αποδοτικότητα των μεθόδων συμπίεσης έγιναν κάποια πειράματα, στα οποία θα αναφερθούμε εκτενέστατα στο 4^o κεφάλαιο. Στο 5^o και τελευταίο κεφάλαιο θα αναφέρουμε τα συμπεράσματα που εξαγάγαμε από τη συγκεκριμένη εργασία αλλά και κάποιους στόχους που έχουν τεθεί για τη συνέχιση αυτής της δουλειάς.

2. ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΧΤΙΣΙΜΑΤΟΣ ΤΟΥ k^2-ΔΕΝΔΡΟΥ ΚΑΙ ΤΩΝ ΠΙΝΑΚΩΝ T ΚΑΙ L

2.1 Πιθανές αναπαραστάσεις του πίνακα γειτνιάσεως

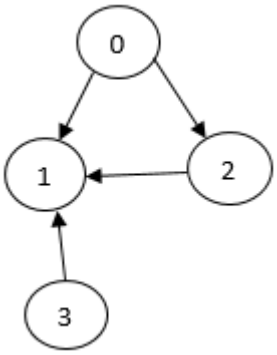
Ο πίνακας γειτνιάσεως (adjacency matrix) [21] μπορεί να αντιπροσωπεύει διάφορα σενάρια όπως για παράδειγμα μπορεί να είναι ο πίνακας ενός γραφήματος ιστού (όπως αναφέρεται και στο πρότυπο άρθρο [1]), όπου κάθε γραμμή και στήλη αναπαριστά μια ιστοσελίδα (Web page). Σε κάθε κελί θα υπάρχει μονάδα αν υπάρχει σύνδεσμος (π.χ. μια ιστοσελίδα να είναι υποσέλιδο μιας άλλης) μεταξύ των ιστοσελίδων και μηδενικό αν δεν υπάρχει. Ένα άλλο σενάριο που θα μπορούσε να αναπαριστά ο πίνακας γειτνιάσεως είναι οι περιοχές ατμοσφαιρικής ρύπανσης μια πόλης, όπου κάθε υποπίνακας του κυρίου πίνακα θα αντικατοπτρίζει μια περιοχή της πόλης. Αν ένας υποπίνακας περιέχει άσσους τότε η αντίστοιχη περιοχή είναι μολυσμένη, αν αποτελείται από μηδενικά τότε δεν υπάρχει μόλυνση κι αν αποτελείται από μηδενικά κι άσσους τότε τα επίπεδα μόλυνσης είναι χαμηλά.

2.2 Δημιουργία του k^2-δένδρου από τον πίνακα γειτνιάσεως

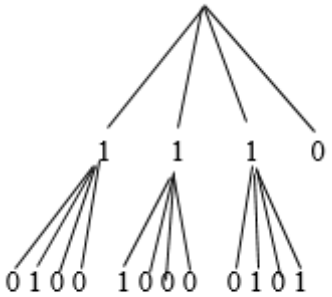
Θα δημιουργήσουμε ένα δένδρο ύψους $h = \lceil \log_k n \rceil$ (με τον όρο ύψος εννοούμε πόσα επίπεδα περιέχει το δένδρο μας [22] και με n συμβολίζουμε το μέγεθος του τετραγωνικού μας πίνακα) από τον πίνακα γειτνιάσεως, που το ονομάζουμε k^2-δένδρο. Ξεκινάμε την αρίθμηση των επιπέδων του δένδρου από το μηδέν, όπου με μηδέν ορίζουμε την ρίζα του δένδρου μας. Κάθε κόμβος περιέχει μονάδα ή μηδενικό (κάθε κόμβος περιέχει 1 bit δεδομένων, κι όχι ένα byte), με μονάδα απεικονίζουμε τους εσωτερικούς κόμβους (internal nodes), δηλαδή τους κόμβους που έχουν έστω ένα παιδί και με μηδενικό τα φύλλα (κόμβοι που δεν έχουν κανένα παιδί) [23]. Το πρώτο επίπεδο του δένδρου μας αποτελείται από τα k^2 παιδιά της ρίζας, τα παιδιά είναι μηδενικά ή μονάδες. Κάθε εσωτερικός κόμβος έχει ακριβώς k^2 παιδιά και τα φύλλα κανένα παιδί.

Αν το n δεν είναι δεν είναι δύναμη του k, τότε επεκτείνουμε τον πίνακάς μας. Δηλαδή προσθέτουμε μηδενικά στο κάτω μέρος και δεξιά του πίνακα μέχρι οι διαστάσεις του να γίνουν κάποια δύναμη του k και το νέο πλάτος του θα είναι $n' = k^h = k^{\lceil \log_k n \rceil}$. Αφού κάνουμε αυτήν την επέκταση (αν είναι απαραίτητη) διαιρούμε τον πίνακά γειτνιάσεως, σύμφωνα με τη μέθοδο MX-Quadtree [24], σε k^2 υποπίνακες του ίδιου μεγέθους, που έχουν k σειρές και k στήλες. Κάθε k^2 υποπίνακας θα είναι παιδί της ρίζας και θα απεικονίζεται με μονάδα αν στα κελιά του υπάρχει μια τουλάχιστον μονάδα ενώ με μηδενικό αν όλα τα στοιχεία του είναι μηδενικά. Για να τοποθετήσουμε τα στοιχεία του δένδρου μας από τον πίνακα γειτνιάσεως ελέγχουμε (αν περιέχουν μηδενικά ή άσσους) τους υποπίνακες από πάνω προς τα κάτω κι από αριστερά προς τα δεξιά. Το πρώτο επίπεδο του δένδρου περιέχει τα παιδιά της ρίζας, κάθε παιδί της που είναι μονάδα έχει και δικά του παιδιά (που τα βρίσκουμε αν διαχωρίσουμε τον υποπίνακα, λόγω του οποίου βάλαμε μονάδα στο παιδί της ρίζας, σε k^2 υποπίνακες) και αναδρομικά γεμίζουμε με μηδενικά κι άσσους. Η αναδρομή τερματίζει όταν κάποιο παιδί είναι 0 ή όταν φτάσουμε στο τελευταίο επίπεδο του δένδρου μας ($h = \log_k n\rceil$).

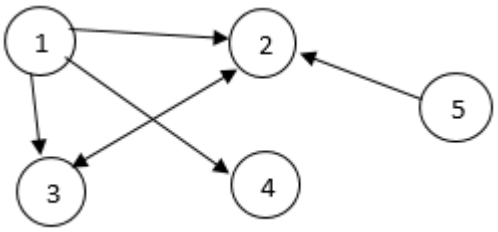
Στην εικόνα 1 απεικονίζεται η αναδρομική διαδικασία που ακολουθούμε για να φτιάξουμε το k^2 δένδρο μας. Στο παράδειγμά μας (εικόνα 1) έχουμε έναν 4 x 4 πίνακα γειτνιάσεως, τον οποίο έχουμε εξάγει από το γράφημα που βρίσκεται αριστερά. Επιπλέον, το k είναι ίσο με 2, άρα το ύψος του δένδρου μας θα είναι $h = \lceil \log_2 4\rceil$ =2. Για να εξάγουμε το πρώτο επίπεδο του δένδρου μας κάνουμε τα παρακάτω βήματα, σπάμε τον αρχικό μας πίνακα σε 2^2 υποπίνακες (και τους ελέγχουμε από αριστερά προς τα δεξιά και από πάνω προς τα κάτω) και βλέπουμε αν οι υποπίνακες περιέχουν στα κελιά τους τουλάχιστον έναν άσσο, αν ναι τότε τοποθετούμε μονάδα στο παιδί της ρίζας. Στο συγκεκριμένο παράδειγμα βλέπουμε πως όλοι οι υποπίνακες περιέχουν τουλάχιστον μία μονάδα εκτός του κάτω δεξιού. Οπότε τα παιδιά της ρίζας θα είναι 1 1 1 0. Για να φτιάξουμε το δεύτερο επίπεδο πάμε στον πρώτο υποπίνακα και τον σπάμε σε 2^2 υποπίνακες άρα το αριστερότερο παιδί της ρίζας θα έχει τα εξής παιδιά 0 1 0 0 (όπως αναφέραμε παραπάνω του πίνακες τους ελέγχουμε από αριστερά προς τα δεξιά κι από πάνω προς τα κάτω). Συνεχίζουμε αυτή τη διαδικασία μέχρι να γεμίσουμε και το τελευταίο επίπεδο του δένδρου μας. Γεμίζουμε τον δένδρο μας από πάνω προς τα κάτω κι από αριστερά προς τα δεξιά.



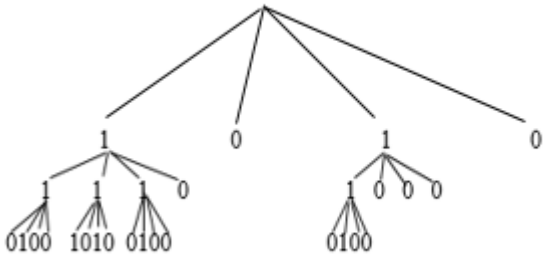
	0	**1**	**2**	**3**
0	0	1	1	0
1	0	0	0	0
2	0	1	0	0
3	0	1	0	0



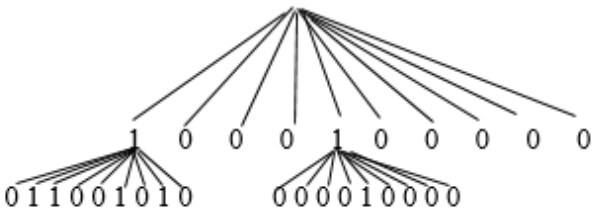
Εικόνα 1: Παράδειγμα δημιουργίας του k^2 – δένδρου από τον πίνακα γειτνιάσεως για k=2 (οι διαστάσεις του πίνακα γειτνιάσεως είναι δύναμη του k, 2^2= 4)



0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



0	1	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0



Εικόνα 2: Παράδειγμα δημιουργίας του k^2 – δένδρου από τον πίνακα γειτνιάσεως για k=2 (πάνω) και για k=3 (κάτω).

2.3 Εξαγωγή πινάκων T και L από το k^2-δένδρο

Η αναπαράσταση του πίνακα γειτνιάσεως γίνεται μέσω του k^2-δένδρου υλοποιείται με τη βοήθεια δύο πινάκων των T και L. Ο πίνακας T (tree) αποτελείται από όλα τα στοιχεία (σε bits) που βρίσκονται στους εσωτερικούς κόμβου του k^2- δένδρου, δηλαδή περιέχει όλα τα επίπεδα του δένδρου εκτός αυτού που βρίσκεται στο τελευταίο επίπεδο ($h = \lceil \log_k n \rceil$) . Αντιθέτως, ο πίνακας L (leaves) αποτελείται από τα στοιχεία (σε bits) που βρίσκονται στο τελευταίο επίπεδο του k^2- δένδρου. Οι πίνακες T και L για το παράδειγμα στην εικόνα 1 είναι T = 1110 και L= 0101 1000 0101. Στην εικόνα 2 βλέπουμε πως οι διαστάσεις του πίνακα γειτνιάσεως δεν είναι κάποια δύναμη του k, οπότε πρέπει να γεμίσουμε τον πίνακά μας με μηδενικά από δεξιά και κάτω ώστε να αλλάξουμε τις διαστάσεις του και να γίνουν κάποια δύναμη του k. Για k = 2 και n = 5 θα γεμίσουμε από δεξιά, τον πίνακά μας, με 3 στήλες μηδενικά και στο κάτω μέρος του με 3 σειρές μηδενικά, ώστε να έχουμε έναν πίνακα 8x8, όπου το 8 είναι δύναμη του k=2. Κι οι πίνακες θα είναι οι εξής: T = 1010 1110 1000 και L = 0100 1010 0100 0100. Για k= 3 και n = 5 θα γεμίσουμε από δεξιά, τον πίνακά μας, με 4 στήλες μηδενικά και στο κάτω μέρος του με 4 σειρές μηδενικά, ώστε να έχουμε έναν πίνακα 9x9, όπου το 9 είναι δύναμη του k=3. Κι οι πίνακες θα είναι οι εξής: T = 1000100000 και L = 011001010 000010000.

Τα στοιχεία των πινάκων μας τα αποθηκεύουμε σε συνεχόμενες θέσεις, κάθε ένα στοιχείο καταλαμβάνει χώρο του ενός bit. Βλέπουμε κι από τα παραδείγματά μας πως για μεγαλύτερα K τα δένδρα μας έχουν λιγότερα επίπεδα αλλά με περισσότερα παιδιά, αναμενόμενο αποτέλεσμα αφού το ύψος το βρίσκουμε από τον τύπο $h = \lceil \log_k n \rceil$. Στο παράδειγμα για k = 2 (εικόνα 2) βλέπουμε ότι τα πρώτα τέσσερα ψηφία (bits) του T αναπαριστούν τους κόμβους 1,2,3 και 4, οι οποίοι είναι παιδιά της ρίζας. Τα επόμενα τέσσερα ψηφία αναπαριστούν τα παιδιά του κόμβου 1. Ο κόμβος 2 δεν έχει παιδιά αφού είναι ίσος με 0. Τα παιδιά του κόμβου 3 ξεκινούν από τη θέση 8 του πίνακα T. Γενικά για να βρούμε το i-παιδί ενός x κόμβου αρκεί να χρησιμοποιήσουμε τον τύπο $\text{rank}(T,x) * k^2+i$. Όπου με τον τύπο rank(T,x) εννοούμε την τάξη του πίνακα T μέχρι κάποια θέση (T[0,x]), δηλαδή το πλήθος των μονάδων που υπάρχουν στον πίνακά μας από την πρώτη θέση (δηλαδή τη θέση 0, αφού η μέτρηση των θέσεων των πινάκων T και L ξεκινάει από το 0) μέχρι τη x. Είναι σημαντικό να επισημάνουμε πως οι εσωτερικοί κόμβοι (internal nodes) ανήκουν μόνο στον πίνακα T. Επιπλέον, πρέπει να αναφερθεί πως το k^2- δένδρο χρησιμοποιείται σαν έννοια για να εξάγουμε τους πίνακες T και L από τον πίνακα γειτνιάσεως, δε θα το κατασκευάσουμε. Αυτά που έχουν ειπωθεί έως τώρα για τη δημιουργία των πινάκων T και L από τον πίνακα γειτνιάσεως θα τα παρουσιάσουμε παρακάτω και στην προγραμματιστική γλώσσα C.

2.3.1 Περιγραφή του αλγορίθμου δημιουργίας των πινάκων T και L

Σε αυτή την παράγραφο θα «χτίσουμε» αναδρομικά το k^2-tree. Θα πραγματοποιήσουμε αναζήτηση του δένδρου σε βάθος [9] (depth-first traversal), από την οποία θα εξάγουμε για κάθε επίπεδο του δένδρου μας έναν πίνακα από bits T_l. Αν είμαστε στο τελευταίο επίπεδο, όπου l=h θα διαβάζουμε τα αντίστοιχα k^2 κελιά. Αν είναι όλα μηδενικά τότε η επιστρεφόμενη τιμή θα είναι 0, αλλιώς παράγουμε τα k^2 bits κι επιστρέφεται η τιμή 1. Αν δεν είμαστε στο τελευταίο επίπεδο κάνουμε k^2 αναδρομικές κλήσεις για τα παιδιά. Αν είναι όλα 0, τότε επιστρέφεται η τιμή 0, αλλιώς επιστρέφεται η τιμή 1. Το αποτέλεσμα κάθε κλήσης αποθηκεύεται ξεχωριστά για κάθε επίπεδο, στον T_l. Ο πίνακας T αποτελείται από όλα τα επίπεδα (δηλαδή όλα τα T_l που έχουμε) του δένδρου μας εκτός του τελευταίου, το οποίο το περιέχει ο πίνακας L.

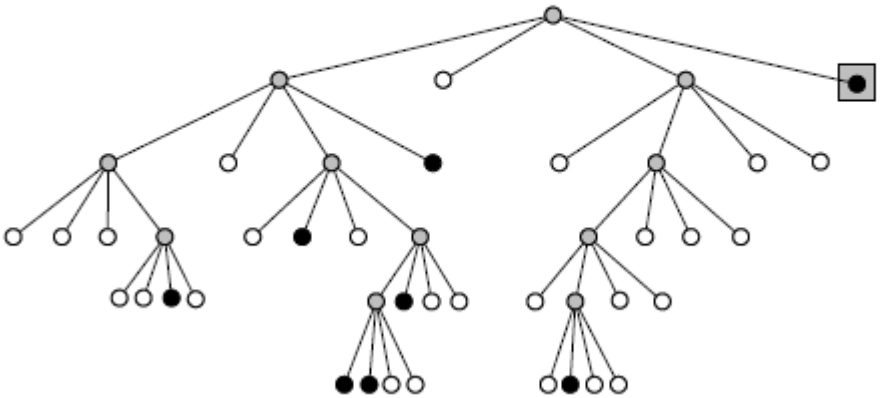
2.3.2 Συμπίεση των μονάδων

Στα παραπάνω παραδείγματα αλλά και στη περιγραφή του αλγορίθμου περιγράφεται συμπίεση μόνο των μηδενικών, δηλαδή όταν υπάρχει ένας k*k υποπίνακας που περιέχει μόνο μηδενικά τότε τον αναπαριστούμε στο δένδρο μας με ένα μόνο μηδενικό (που δεν έχει παιδιά). Στο άρθρο Compact Querieable Representations of Raster Data [2] περιγράφεται κι η συμπίεση στις μονάδες, δηλαδή όταν έχουμε έναν υποπίνακα γεμάτο με μονάδες τότε μπορεί να αναπαρασταθεί στο δένδρο μας με έναν και μόνο άσσο, ο οποίος δεν θα έχει κανένα παιδί. Στην εργασία πραγματοποίησα και τις δύο μεθόδους συμπίεσης. Όπως βλέπουμε και στην εικόνα 4 ο τελευταίος υποπίνακας είναι γεμάτος με άσσους, οπότε αναπαρίσταται στο δένδρο μας (όπως βλέπουμε και στο δένδρο αριστερά στην εικόνα 4) με μια μονάδα. Είναι πολύ σημαντικό να αναφέρουμε πως αρχικά ο κώδικας ήταν χτισμένος με τέτοιον τρόπο ώστε να γίνεται συμπίεση μόνο των μηδενικών και στη συνέχεια εξελίξαμε τον κώδικα έτσι ώστε να γίνεται συμπίεση και στους άσσους.

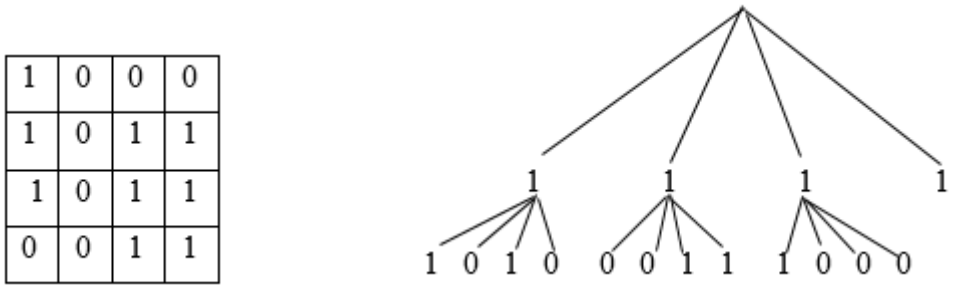
Το δένδρο μας θα περιέχει κόμβους με τρία πιθανά «χρώματα» μαύρο, άσπρο και γκρι (τα χρώματα είναι θεωρητικά, ώστε να διακρίνουμε από τι παιδιά αποτελείται κάθε κόμβος δηλαδή μηδενικά ή άσσους). Οι άσπροι κόμβοι είναι περιοχές με μηδενικά, οι μαύροι κόμβοι είναι περιοχές με μονάδες κι οι γκρι κόμβοι είναι εσωτερικοί κόμβοι που αποτελούνται από μηδενικά κι άσσους. Με αυτή τη μέθοδο μπορούμε να αποφύγουμε να αναπαραστήσουμε στο δένδρο μας μεγάλες περιοχές με μονάδες. Στο συγκεκριμένο άρθρο περιγράφονται δύο τρόποι συμπίεσης των μονάδων η πρώτη λειτουργεί καλύτερα όταν το πλήθος των μονάδων και των άσσων στον πίνακά μας δε διαφέρει και

πολύ (2-bits Variant), ενώ η δεύτερη όταν το πλήθος τους είναι αρκετά διαφορετικό (Unbalanced (1-5)-bits Variant). Σε αυτή την εργασία χρησιμοποιήθηκε η δεύτερη μέθοδος.

Σε αυτή τη μέθοδο χρησιμοποιούμε τους πίνακες T και L όπως τους περιγράψαμε παραπάνω με μια παραλλαγή, κάθε φορά που βρίσκουμε έναν υποπίνακα (στον πίνακα γειτνιάσεως) «γεμάτο» με μονάδες τον αναπαριστούμε με μια μόνο μονάδα κι ως παιδιά του τοποθετούμε K^2 μηδενικά. Στην εικόνα 3 βλέπουμε ένα παράδειγμα συμπίεσης των μονάδων, σύμφωνα με την παραπάνω περιγραφή ο πίνακας T θα είναι ο εξής : T=1011 1011 0100 0000 0001 0101 0000 1000 0010 0000 1100 0100. Βλέπουμε για παράδειγμα ότι ο δεξιότερος κόμβος, που είναι μαύρος (δηλαδή αποτελείται από μονάδες) αναπαρίσταται στον πίνακα T (στη θέση 3 του T, αν ξεκινήσουμε τη μέτρηση των θέσεων του T από το 0) με μία μονάδα και για παιδιά του έχει 4 μηδενικά (θέσεις από 12 έως 15). Γενικά όταν κάνουμε συμπίεση στις μονάδες θα πρέπει, σε κάθε αναζήτηση σου δένδρου μας, να λαμβάνουμε υπόψιν τα k^2-1 αδέρφια [25] του εκάστοτε κόμβου, ώστε να ξέρουμε αν ο κόμβος μας είναι άσπρος (δηλαδή το συγκεκριμένο bit είναι 0 αλλά έστω ένα από τα αδέρφια του είναι 1) ή αν είμαστε σε περιοχή με μονάδες (δηλαδή το τρέχον bit είναι 0 και όλα τα αδέλφια του είναι 0, που σημαίνει ότι ο γονικός κόμβος ήταν πραγματικά μαύρος), όπως χαρακτηριστικά μας δείχνει και το παρακάτω παράδειγμα. Επομένως, κάθε φορά που συναντάμε k^2 μηδενικά στον πίνακα T ξέρουμε πως ο γονιός αυτών θα είναι μονάδα. Δηλαδή οι μαύροι κόμβοι αντιμετωπίζονται ως γκρι με k^2 άσπρα παιδιά.



Εικόνα 3: Παράδειγμα απεικόνισης των διάφορων κόμβων ανάλογα με το περιεχόμενο των παιδιών του.



Εικόνα 4:Παράδειγμα συμπίεσης των μονάδων, για k=2.

3. ΥΛΟΠΟΙΗΣΗ ΤΟΥ k^2-ΔΕΝΔΡΟΥ ΚΙ ΕΦΑΡΜΟΓΗ ΕΡΩΤΗΜΑΤΩΝ

3.1 Γενικά

Όπως έχουμε αναφέρει παραπάνω η συμπίεση γραφημάτων κρίνεται απαραίτητη αφού το πλήθος των στοιχείων που απαιτούν επεξεργασία έχει αυξηθεί δραματικά λόγω της ραγδαίας εξέλιξης της τεχνολογίας τα τελευταία χρόνια. Όμως μια μέθοδος συμπίεσης δε είναι αρκετά αποτελεσματική αν καταφέρνει μόνο να καταλαμβάνει λιγότερο χώρο στην κύρια μνήμη, μας ενδιαφέρει κι ο χρόνος που καταναλώνεται για την υλοποίηση αυτής αλλά και κάποιων ερωτημάτων που θέτουμε. Στις παρακάτω παραγράφους θα αναφερθούμε εκτενέστερα στον κώδικα για την υλοποίηση της συμπίεσης αλλά και των διάφορων ερωτημάτων. Αρχικά είναι χρήσιμο να αναφέρουμε πως έχουμε ορίσει ένα struct (compressed_graph) το οποίο σαν πεδία περιέχει όλα τα στοιχεία που χρειαζόμαστε ώστε να είναι δυνατή η δημιουργία πολλών δένδρων με το ίδιο κώδικα, απλά βάζοντας διαφορετικές τιμές για το k και το n (αρχικό μέγεθος του πίνακα). Επίσης, πολλές μεταβλητές που περιέχει τις χρησιμοποιούμε σε πολλά μέρη του κώδικα, όπως για παράδειγμα τις συναρτήσεις κι επειδή δεν ήταν πρακτικό να βάλουμε πολλές μεταβλητές σαν καθολικές (global) τις ενσωματώσαμε όλες σε ένα struct, το οποίο βλέπουμε παρακάτω.

```
typedef struct compressed_graph{
    int upsos;
    unsigned char *T;/*edw orizw tous duo telikous pinakes T kai L*/
    unsigned char *L;
    int T_size;/*se auth thn metaviliti tha apothikeuetai to megethos tou T*/
    int *acc;/*se auton ton pinaka tha apothikeuoyme ana x stoixeia to plithos
    twn monadwn ston pinaka T */
    int K;
    int N_new;
    int X;
    int **pinakas_geitniasews;
    int **pinakas_geitniasews_initial;
    //dhlwsh kai arxikopoihsh ths kefalis gia thn lista twn successors
    node_T * s_head;
    //dhlwsh kai arxikopoihsh ths kefalis gia thn lista twn successors
    node_T * p_head;
    //kefali gia hn lista poy ftiaxnoume sthn range,kathe komvos tha periexei
    //duo times
    node_T *head_dp_dq;
    }compressed_graph_T;
```

Επίσης, είναι σημαντικό να αναφερθεί το γεγονός πως για την απλοποίηση της μορφής του κώδικα, τον «σπάσαμε» σε 3 κομμάτια.

- Στο πρώτο κομμάτι περιέχονται όλα τα πρότυπα των συναρτήσεων που χρησιμοποιήσαμε για την υλοποίηση του κώδικά μας κι ονομάστηκε comp.h (βιβλιοθήκη των συναρτήσεων μας) και καλείται στα άλλα δύο κομμάτια που θα αναφέρουμε παρακάτω ώστε να γίνεται η σύνδεση μεταξύ των τριών κομματιών.
- Στο δεύτερο κομμάτι περιέχονται όλες οι συναρτήσεις κι η λειτουργία τους κι ονομάστηκε comp.c.
- Το τρίτο και τελευταίο κομμάτι περιλαμβάνει τη συνάρτηση main κι ως επακόλουθο ονομάστηκε main.c.

Παρακάτω βλέπουμε και τη διαδικασία που ακολουθήθηκε για να συμπεριληφθεί το ένα κομμάτι κώδικα στο άλλο. Αλλά κι ήδη υπάρχουσες βιβλιοθήκες που χρησιμοποιήθηκαν.

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <sys/time.h>

#include "comp.h"
```

Comp.h

```
#ifndef COMP_H_INCLUDED
#define COMP_H_INCLUDED
```

Comp.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <sys/time.h>

#include "comp.h"
```

Επιπροσθέτως, όπως έχουμε αναφέρει παραπάνω στον πίνακα T αποθηκεύουμε τα στοιχεία σε bits, οπότε σε όλες τις συναρτήσεις (θα αναφερθούμε εκτενέστατα παρακάτω σε αυτές) όταν κάνουμε αναζήτηση στα στοιχεία του T πρέπει να γίνεται σε επίπεδο bits κι όχι bytes. Για να μην επαναλαμβάνουμε κάθε φορά τις κατάλληλες πράξεις για την αναζήτηση σε bits δημιουργήσαμε την μακροεντολή: `#define H(P,l) (P[l/8]&one_mask[l%8]?1:0)`. Δηλαδή με αυτή τη μακροεντολή γίνεται ο υπολογισμός της θέσης μας στον πίνακα από bits ενώ μας δίνεται η θέση στον πίνακα από bytes.

3.2 Παρουσίαση της συνάρτησης Build

Σε αυτή την παράγραφο θα παρουσιαστεί και θα επεξηγηθεί η συνάρτηση με την οποία αναδρομικά από τον πίνακα γειτνιάσεως χτίζουμε το k^2- δένδρο και κατά συνέπεια τον πίνακα T.

```
/*Me th sunartisi build xtizoume ton pinaka T apo ton pinaka geitniasews alla
kai ton T_init,o opoios periexei ta stoixeia tou
dendrou mas (kathe epipedo tou dendrou mas anaparistatai se mia grammh tou T_init).
Epishs me ton parakatw kwdika ftiaxnoume ton
T_filled*/
int build(compressed_graph_T * G,int submatrix_width,int current_level,int row,int col, int **T_init,int *T_filled){
    int i,j,C[G->K*G->K],flag;
    int count=0;

/*Ypologismos tou vohthitikou pinaka C[i],diladi o pinakas poy tha proskolithei
ston T.O pinaka C[i] periexei se kathe anadromi
K*K stoixeia ,pou kathe fora an den einai ola ta stoixeia 0 h 1 (otan leme oti
ola ta stoixeia einai 0 h 1 ennooume pws enas
upopinakas apoteleitai mono apo monades h assous) ta topothetoume ston pinaka T*/
    for(i=0;i<G->K;i++){
        for(j=0;j<G->K;j++){
            if(current_level==G->upsos){/*an eimaste sto teleytaio epipedo tou
                    dendrou (to dendro xrhsimopoieitai san ennoia)*/
                C[count]=G->pinakas_geitniasews[row+i][col+j];
                count++;/*metritis gia na kseroume se poia thesi eimaste ston C*/
            }
            else{
                C[count]=build(G,submatrix_width/G->K,current_level+1,
                               row+i*(submatrix_width/G->K),
                               col+j*(submatrix_width/G->K),T_init,T_filled);
                count++;
            }
        }
    }
```

Η συνάρτηση build έχει ως παραμέτρους μια μεταβλητή G τύπου compressed graph, το μέγεθος του πίνακα γειτνιάσεως (submatrix width), το τρέχων επίπεδο του δένδρου (current_level), την τρέχουσα σειρά (row) και την τρέχουσα στήλη (col) που βρισκόμαστε σε κάθε αναδρομική κλήση. Τέλος, περιέχει δύο πίνακες τους T_init και T_filled. Ο πίνακας T_init περιέχει όλα τα στοιχεία του δένδρου μας ανά επίπεδο και στον T_filled καταγράφεται το πλήθος των στοιχείων του κάθε επιπέδου του k^2-δένδρου. Όπως βλέπουμε παραπάνω στον κώδικα για να δημιουργήσουμε τον πίνακα T χρησιμοποιούμε ένα βοηθητικό πίνακα τον C. Σε κάθε αναδρομική κλήση της συνάρτησης ο πίνακας C περιέχει k^2 στοιχεία, τα οποία υπόκεινται σε κάποιους ελέγχους τους οποίους θα δούμε παρακάτω αναλυτικά. Αυτά τα k^2 στοιχεία τοποθετούνται ανά επίπεδο στον πίνακα T_init από τον οποίο στη συνέχεια θα χτίσουμε τον πίνακα T. Η αναδρομική σχέση «σπάει» τον πίνακά μας σε υποπίνακες (όπως επεξηγήσαμε παραπάνω) ώστε να φτάσει στο τελευταίο επίπεδο του δένδρου μας, όπου περιέχει

σχεδόν όλα τα στοιχεία του πίνακάς μας. Κάθε φορά που ο πίνακας C περιέχει K^2 στοιχεία, τα οποία δεν είναι μηδενικά ή άσσοι τα τοποθετούμε στον πίνακα T_init.

Όπως βλέπουμε παρακάτω έχουμε ορίσει μια μεταβλητή flag, η οποία παίρνει την τιμή 0 αν όλα τα στοιχεία μας στον C είναι μηδενικά (δηλαδή ένας υποπίνακάς μας είναι γεμάτος με μηδενικά) ή την τιμή 1 αν υπάρχει έστω και μία μονάδα. Όταν η μεταβλητή flag πάρει την τιμή 0, τότε τερματίζει η μια κλήση της συνάρτησής μας κι επιστρέφεται η τιμή 0. Με αυτό τον τρόπο καταφέραμε να συμπιέσουμε τον πίνακά μας, δηλαδή να αναπαριστούμε με ένα μόνο μηδενικό έναν ολόκληρο υποπίνακα με μηδενικά (συμπίεση στα μηδενικά). Επειδή εμείς θέλουμε να κάνουμε και συμπίεση στις μονάδες πρέπει να πάρουμε κι άλλους περιορισμούς για τον πίνακα C. Αν είμαστε στο τελευταίο επίπεδο του δένδρου μας κι όλα τα k^2 στοιχεία του C είναι μονάδες (δηλαδή ένας υποπίνακας περιέχει μόνο μονάδες), τότε η μεταβλητή flag παίρνει την τιμή -1.Τερματίζει η μια κλήση της συνάρτησής μας κι επιστρέφεται η τιμή -1, εάν υπάρχει έστω ένα μηδενικό τότε τιμή flag παίρνει την τιμή 1 και συνεχίζεται η υπόλοιπη διαδικασία. Επίσης, υπάρχει η πιθανότητα όλα τα στοιχεία του C να είναι ίσα με -1, τότε θέτουμε τη μεταβλητή flag ίση με -1, αλλιώς τη θέτουμε ίση με 1, όπως βλέπουμε παρακάτω. Αν όλα τα στοιχεία του C είναι -1, τότε τα απεικονίζουμε με ένα και μοναδικό -1 (συμπίεση στους άσσους).

```
/*an ola ta K*K stoixeia tou C[i] einai diafora tou 0 tote h metavliti flag
 pairnei thn timh 1,an flag==0 tote shmainei pws ola ta stoixeia mas einai
 mhdenika kai h metavliti flag pairnei thn timh 0,se auth thn periptwsh to
 programma mas termatizei kai mas epistrefetai h timh 0. Auto sumvainei
 epeidi giati theloume na anaparasthsoume enan upopinaka  gemato me 0 me
 monoena mhdeniko*/
flag=0;
for(i=0;i<G->K*G->K;i++){
    if( C[i]!=0)
       flag=1;
}
if (flag == 0) {
    return 0;
}
/*Me ton parakatw kwdika kanoume elegxo gia to an ena upodendro einai gemato
monades.An eimaste sto teleutaio epipedo tou dendrou mas ki estw ena apo ta
stoixeia tou C[i] einai 0 (ton elegxo gia to an einai ola mhdenika ton
kaname parapanw) tote h metavliti flag mas pairnei thn timh 1,an flag=-1
tote shmainei pws ola ta stoixeia mas einai monades kai h metavliti flag
pairnei thn timh -1 ki epistrefetai h timh -1,giati theloume theloume ena
upodendro me monades na anaparistatai mono me mia monada.Otan ena upodendro
einai gemato me 1 tote h epistrefomenh timi tha einai -1*/
flag = -1;
if(current_level==G->upsos){
    for(i=0;i<G->K*G->K;i++){
        if(C[i]!=1){/*se periptwsh pou ola ta K*K stoixeia tou C den einai
                1 tote flag=1 */
            flag=1;
        }
    }
    if (flag == -1) {/*an h metavlititi flag den exei thn timi 1 tha exei
            thn timi -1 ,auto shmainei pws
            ola ta K*K stoixeia mas tha einai 1 ,opote epistrefomeni timi
            tha einai h -1*/
       return -1;
    }
}

/*Prepei na kanoume ki elegxo an ola ta K*K stoixeia mas einai -1.
 An den eimaste sto teleutaio epipedo ,h metavliti flag tha einai
 ish me 1 otan ola ta K*K stoixeia den einai -1.
 Termatizei h anazhthsh otan h metavliti flag pairnei tin timh -1
 (dhladh ola ta stoixeia mas einai -1)
 kai mas epistrefetai h timh -1*/
 else {
    flag = -1;
    for(i=0;i<G->K*G->K;i++){
        if(C[i]!=-1)/*se periptwsh pou ola ta K*K stoixeia tou C
            den einai -1 tote h epistrefomeni timh tha einai 1 */
          flag=1;
    }
    if (flag == -1)/*se periptwsh pou einai ola ta stoixeia einai -1
        ,h epistrefomenh timh tha einai -1*/
        return -1;
 }
```

Τελικά, μετά τους ελέγχους που πραγματοποιούμε, τοποθετούμε τα στοιχεία μας στον T_init (σε κάθε γραμμή του περιέχει τα στοιχεία του δένδρου μας ανά επίπεδο, οπότε θα έχει τόσες γραμμές όσο το ύψος του δένδρου και στήλες όσα τα στοιχεία του τελευταίου επιπέδου του δένδρου) και ταυτόχρονα έχουμε έναν άλλον πίνακα τον T_filled στον οποίο αποθηκεύεται κάθε φορά το πλήθος των στοιχείων που περιέχει κάθε γραμμή του T_init. Όπως βλέπουμε και παρακάτω.

```
/*T_init einai o pinakas pou apothikeuontai ta stoixeia tou C ana
epipedo tou dendrou(kathe epipedo mia grammi).O T_filled einai enas
 voithitikos pinakas pou mas leei kathe fora apo poio shmeio ki epeita
 prepei na gemisoume ton T_init,diladh ana K*K stoixeia o T_filled tha
 auksanetai kata K*K */
for(i=0;i<G->K*G->K;i++){
    T_init[current_level-1][T_filled[current_level-1]++]=C[i];
 }
```

3.3 Λειτουργία της συνάρτησης Create Graph

Σε αυτή τη συνάρτηση δημιουργούμε τους πίνακες T και L με τη βοήθεια των πινάκων που δημιουργήθηκαν από τη συνάρτηση Build, T_filled και T_init. Στη συνάρτηση Create Graph καλούμε τη συνάρτηση Build(G, pow(G->k,G->upsos),1,0,0,T_init,T_filled). Η πρώτη παράμετρος G είναι μια μεταβλητή τύπου compressed_graph, με τη μεταβλητή pow(G->k,G->upsos) βρίσκουμε το μέγεθος του πίνακα γειτνιάσεως, ακόμη και μετά την επέκταση των μηδενικών, η επόμενη παράμετρος είναι το ύψος του δένδρου που θα έχει ως αρχική τιμή το 1 (αφού έχουμε ορίσει η ρίζα του δένδρου έχει την τιμή μηδέν και το πρώτο επίπεδο την τιμή 1), μετά ακολουθούν οι τιμές 0 και 0 (που είναι οι θέσεις (0,0) του πίνακα γειτνιάσεως, αφού ξεκινάμε από την πρώτη θέση του πίνακά μας) και τέλος έχουμε τις παραμέτρους T_init και T_filled.

```
build(G,pow(G->K,G->upsos),1,0,0,T_init,T_filled);/*to pow(k,h)
einai to megethos tou pinaka*/

/*edw kanoume antikatastash ta -1 me assoous kai h metavliti rank metraei
tous assous pou exoume (afou kanoume antikatastash
ta -1 me 1 h metavliti rank tha auksanetai ki otan vriskoume -1). Se auth
thn sunarthsh pame kai vazoyme sta paidia kathe -1
K*K mhdenika, gia na vroume se poia thesi vriskontai ta paidia arkei na
kanoume thn praksi rank*(K*K). */
for(i=0;i<G->upsos;i++){
    rank =0;/*opou rank einai mia metavliti pou krataei to plithos twn
    asswn kai twn -1 se kathe grammi tou T_init ,prepei
    na kseroume pou o T_init periexei -1 kai 1.*/
    for(j=0;j<T_filled[i];j++)
        if (T_init[i][j] == 1)
            rank++;
        else if (T_init[i][j] == -1){
            rank++;
            for (k =T_filled[i+1]-1; k> (rank-1)*(G->K*G->K)-1;k--)
                /*edw prepei na kanoume ta stoixeia mas metskinish pros
            ta aristera gia na valoume ta K*K mhdenika mas
            (diladi ta paidia kathe -1),opote prepei na pame sto akrivws
            apo katw level otou T_init apo auto pou eimaste (giati ta paidia
```

```
                kathe komvou vriskontai sto ameswS epomeno level
                apo auto pou vrisketai o komvos mas )kai na topothethsoume K*K
                 mhdenika sta paidia tou -1 */
                    T_init[i+1][k+G->K*G->K] = T_init[i+1][k];

                for ( l= rank*(G->K*G->K)-(G->K*G->K);l<rank*(G->K*G->K);l++)
                    /*edw topothetoume ta mhdenika sta paidia twn -1,
                    epeidh tha eimaste se allo epipedo apo tous goneis the prepei
                     na afairesoume tous K*K goneis apo to
                    epomeno epipedo*/
                      T_init[i+1][l] = 0;
                T_filled[i+1]+=G->K*G->K;
                T_init[i][j] = 1;
        }
}
```

Στη συνάρτηση Build σε περίπτωση που έχουμε έναν ολόκληρο υποπίνακα με άσσους τον αναπαριστούσαμε με -1 στον T_init. Εμείς όμως θέλουμε ο πίνακας T_init να περιέχει μόνο μηδενικά κι άσσους (αφού από αυτόν θέλουμε να φτιάξουμε τους T και L που περιέχουν μόνο μηδενικά κι άσσους). Για αυτό τον λόγο δημιουργήσαμε την παραπάνω επανάληψη, στην οποία κάθε φορά που συναντάμε -1 στον T_init το κάνουμε 1 και πάμε στο επόμενο επίπεδο (δηλαδή στο επίπεδο που βρίσκονται τα παιδιά του -1 που συναντήσαμε), από αυτό που βρίσκεται το -1, και τοποθετούμε k^2 μηδενικά. Επιπλέον, κάθε φορά που τοποθετούμε k^2 μηδενικά σε ένα επίπεδο του T_init πρέπει τα στοιχεία μας, που βρίσκονται σε αυτό το επίπεδο, να ολισθαίνουν k^2 θέσεις δεξιά ώστε τα παιδιά του κόμβου που περιέχει το -1 να βρίσκονται στην κατάλληλη θέση. Ως αντίκτυπο αυτού πρέπει κάθε φορά να προσθέτουμε στον T_filled k^2 στοιχεία στο ανάλογη γραμμή. Χρησιμοποιούμε και μια βοηθητική μεταβλητή rank, η οποία «κρατάει» το πλήθος των 1 και των -1 σε κάθε επίπεδο του T_init. Για να γίνει πιο κατανοητή η παραπάνω διαδικασία θα αναφερθούμε σε ένα παράδειγμα με k=2. Έστω ότι συναντάμε στην πρώτη γραμμή, στη δεύτερη θέση του T_init ένα -1, και πριν από αυτό υπάρχει μια μονάδα, τότε το (το -1) αντικαθιστούμε με 1 και πάμε στην επόμενη γραμμή του πίνακα και βάζουμε στη θέση 4, τέσσερα μηδενικά κι όλα τα υπόλοιπα στοιχεία του ίδιου επιπέδου τα τοποθετούμε 4 θέσεις δεξιά (στον πίνακα T_init ξεκινάμε από τη θέση 0,0). Με αυτή τη διαδικασία θα αλλάξει κι ο πίνακας T_filled, δηλαδή θα αυξήσουμε κατά 4 μονάδες το στοιχείο που αντιστοιχεί στη γραμμή που υπέστη την αύξηση των στοιχείων της.

Όπως αναφέρθηκε και πιο πάνω το κάθε κελί ενός πίνακα καταλαμβάνει χώρο ενός byte στη μνήμη του υπολογιστή μας, αλλά εμείς θέλουμε κάθε κελί να καταλαμβάνει κάθε κελί 1 bit. Για αυτό τον λόγο δεσμεύουμε χώρο σε bits για τους πίνακες T και L. Επιπλέον, στο παρακάτω κομμάτι κώδικα βρίσκουμε το μέγεθος του πίνακα T (T_size), το οποίο για να το βρούμε αρκεί να προσθέσουμε τα στοιχεία του T_filled εκτός αυτού που αντιπροσωπεύει το πλήθος των στοιχείων που περιέχει το

τελευταίο επίπεδο του δένδρου μας. Ο πίνακας T_filled είναι ένα μονοδιάστατος πίνακας που έχει τόσες στήλες όσο είναι και το ύψος του δένδρου μας.

```
 /*T_size einai to plithos olwn twn stoixeiwn tou T,ki auto to vriskoume
  an prosthesoume ola ta stoixeia
(ektos tou teleutaiou stoixeiou pou einai o pinakas L) tou T_filled*/
G-> T_size =0; /*edw arxikopoioume to T_size pou exoume orisei */
for(i=0;i<G->upsos-1;i++){
    G->T_size =G->T_size+ T_filled[i];
}

 /*desmeusi xwrou gia ton T se bits,edw diairw me to 8 giati 1 byte=8 bits
 ki o upologistis apothikeuei se bytes
.Emeis theloume ta stoixeia twn pinakwn mas na mhn pianoun xwrou ena
byte(megethos char)alla ena 1 bit*/
G->T=(unsigned char*)malloc(ceil(G->T_size/8.0));
 if(G->T==NULL){
    printf("Error in G->T");
    exit(1);
 }
/*desmeusi xwrou gia ton L se bits*/
 G->L=(unsigned char*)malloc(ceil(T_filled[G->upsos-1]/8.0));
 if(G->L==NULL){
    printf("Error in G->L");
    exit(1);
 }
T_pos =0;/*arxikopoihsh tou metriti mas*/

/*edw ftiaxnoume ton teliko pinaka T kai topothetoume ta stoixeia mas
 se enan pinaka apo bits,dhladh
pairnoume 8 stoixeia apo ton T_init(sthn arxh kathe stoixeio pianei ena byte)
kai ta topothetoume se ena byte
(dhladh theloume kathe stoixeio mas na pianei 1 bit)*/
for (i = 0; i<G->upsos-1; i++) {
    for (j =0; j < T_filled[i]; j++){
        if (T_init[i][j] == 1){/*an to stoixeio mas ston T_init einai 1 tote
                arkei na pollaplasiasoume me thn katallhlh
        maska wste na mpei sthn katallhlh thesi monada kai ola ta prohgoumena
        na meinoun idia */
           G->T[(int)(ceil((j+T_pos+1)/8.0)-1)] |= (unsigned char)one_mask[(j+T_pos)%8];
        }
        else{
            G->T[(int)(ceil((j+T_pos+1)/8.0)-1)] &= (unsigned char)zero_mask[(j+T_pos)%8];
        }
        number = G->T[(int)(ceil((j+T_pos+1)/8.0)-1)];
    }
    T_pos=T_pos+ T_filled[i];
}
for (j = 0; j< T_filled[G->upsos-1]; j++) {
    if (T_init[G->upsos-1][j] == 1){
       G-> L[(int)(ceil((j+1)/8.0)-1)] |= one_mask[j%8];
    }
    else{
       G-> L[(int)(ceil((j+1)/8.0)-1)] &= zero_mask[j%8];
    }
}
```

Παραπάνω βλέπουμε τη διαδικασία μετατροπής του χώρου που καταλαμβάνουν οι πίνακες T και L από bytes σε bits, κάθε byte ισούται με 8 bits. Για να κάνουμε τη διαδικασία μετατροπής θα χρησιμοποιήσουμε δύο πίνακες (zero_mask[8]={0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F} και one_mask[8] ={0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80}), που περιέχουν κάποιες

βοηθητικές μάσκες τις οποίες θα χρησιμοποιούμε ανάλογα με το αν ο T_init περιέχει μηδενικό ή μονάδα [26]. Δηλαδή αυτό που θέλουμε να καταφέρουμε με αυτή τη διαδικασία είναι να «χωρέσουμε» οχτώ ψηφία (bits) σε ένα κελί του πίνακά μας, όπου κάθε κελί του καταλαμβάνει χώρο του ενός byte. Για να γίνει αυτό πρέπει να κάνουμε τις κατάλληλες πράξεις ώστε όλα τα ψηφία (bits) να μπουν στην κατάλληλη θέση σε κάθε byte. Σαν πρώτο βήμα πρέπει να κάνουμε έλεγχο για το αν τα στοιχεία του T_init είναι μονάδες. Αν όντως κάποιο είναι μονάδα τότε κάνουμε μια λογική πρόσθεση με την κατάλληλη μάσκα (δηλαδή τη μάσκα που περιέχει μονάδα στο ίδιο bit με το bit που υπάρχει μονάδα στο byte που ελέγχουμε) και τοποθετούμε το στοιχείο μας στην κατάλληλη θέση του πίνακα T από bits.Αν όμως το στοιχεία μας είναι μηδενικό, τότε κάνουμε λογικό πολλαπλασιασμό με την κατάλληλη μάσκα.

Για να κατανοηθεί καλύτερα η μέθοδος θα παρουσιάσουμε ένα παράδειγμα. Αν στον πίνακα από bytes (ξεκινώντας τη μέτρηση από το 0) το στοιχείο μας βρίσκεται στη θέση 53, τότε στον πίνακα από bits θα βρίσκεται στο $\lceil 53/8.0 \rceil - 1$=6 byte (αριθμώντας τα bytes από το 0) και συγκεκριμένα στο bit 53mod8=5 (αριθμώντας από το 0 τα bits σε κάθε byte). Δηλαδή το στοιχείο, στον πίνακα από bytes, στη θέση 53 βρίσκεται στο πέμπτο bit του έκτου byte στον πίνακα από bits.

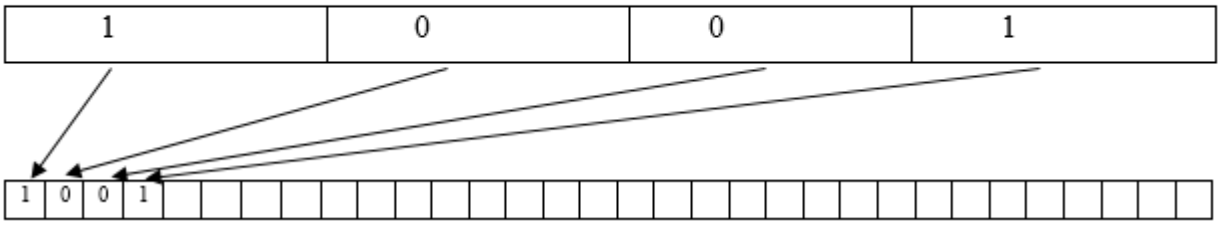
- Στο Byte 0 θα περιέχονται τα bits των θέσεων 0-7
- Στο Byte 1 θα περιέχονται τα bits των θέσεων 8-15
- Στο Byte 2 θα περιέχονται τα bits των θέσεων 16-23

  .

  .

  .
- Στο Byte 6 θα περιέχονται τα bits των θέσεων 48-55

Επίσης, θα αναφερθώ σε ένα παράδειγμα για την χρήση των μασκών. Αν έχουμε έναν πίνακα από bytes και στην πρώτη θέση περιέχει μια μονάδα, στις άλλες εφτά συνεχόμενες θέσεις περιέχει μηδενικά και μονάδες, κι εμείς θέλουμε να τοποθετήσουμε τα στοιχεία του σε έναν πίνακα από bits τότε για να τοποθετήσουμε το πρώτο στοιχείο στον πίνακα από bits αρκεί να κάνουμε μια λογική πρόσθεση με τη μάσκα 1 0 0 0 0 0 0 0, ώστε το πρώτο στοιχείο στον πίνακα από bits να γίνει ένα και τα υπόλοιπα να παραμείνουν ως έχουν. Και συνεχίζουμε την ίδια διαδικασία για όλα τα στοιχεία του πίνακα από bytes. Παρακάτω θα παρουσιάσουμε μια σχηματική απεικόνιση, όπως βλέπουμε 8 bits αντιστοιχούν σε 1 byte.



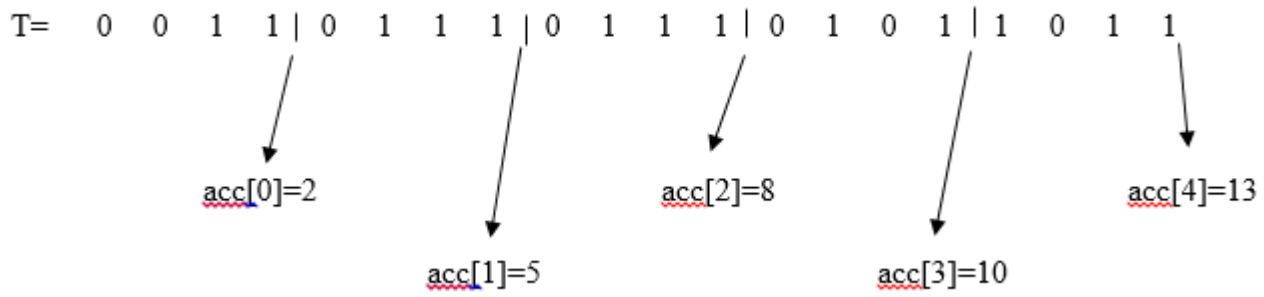
Εικόνα 5: Παράδειγμα αντιστοίχισης των στοιχείων ενός πίνακα από bytes (πάνω) σε έναν πίνακα από bits (κάτω).

```
G->acc=(int*)malloc(sizeof(int)*(G->T_size/G->X));/*desmeusi xwrou gia ton
pinaka acc ana X theseis*/
if(G->acc==NULL){
    printf("Error in G->acc");
    exit(1);
 }

rank_without_limits=0;
metritis=0;
/*edw ftiaxnouume ton pinaka acc dhladh ton pinaka pou ana x theseis kratame
 thn taksi enos pinaka(auto to kanoume
 gia na kanoume kathe fora ligoterous upologismous)*/
for(i=0;i<G->T_size;i++){
    if(H(G->T,i)){/*upologismos twn monadwn ston pinaka T*/
        rank_without_limits++;
    }
    if((i+1)%G->X==0){/*otan eimaste se theseis pollaplsia tou X tote pame
        ston acc kai topothetoume tous assous se authn thn thesi*/
        G->acc[metritis]=rank_without_limits;
        metritis++;/*O metritis autos krataei tis theseis tou acc*/
    }
}
```

Στην συνάρτηση Create Graph περιέχεται κι η δημιουργία του πίνακα acc (παραπάνω βλέπουμε την υλοποίησή του σε C), που είναι ένας μονοδιάστατος πίνακας τον οποίο φτιάχνουμε με τη βοήθεια του πίνακα (παραπάνω βλέπουμε και την υλοποίηση αυτού του πίνακα σε C). Ο πίνακας acc είναι ένας βοηθητικός πίνακας που αποθηκεύουμε ανά X θέσεις (τιμή για το X δίνει ο χρήστης στην αρχή του προγράμματος) την τάξη του πίνακα T, με τον όρο τάξη εννοούμε το πλήθος των μονάδων από τη θέση 0 του πίνακα T μέχρι κάποια συγκεκριμένη θέση. Αυτός ο πίνακας θα μας βοηθήσει στον υπολογισμό τη τάξης μέχρι όποια θέση θέλουμε στον πίνακα T. Η αρίθμηση των θέσεων του ξεκινάει από το 0. Στον παραπάνω κώδικα βλέπουμε τη δημιουργία του αλλά και τη

δέσμευση χώρου για αυτόν τον πίνακα. Ο χώρος που καταλαμβάνει θα είναι ίσος με το συνολικό μέγεθος του πίνακα T (T_size) διαιρούμενο με το X. Αν για παράδειγμα έχουμε έναν πίνακα T=0 0 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 1 και θέσουμε το X ίσο με 4, τότε ο πίνακας acc θα είναι ίσος με acc={ 2, 5, 8, 10, 13}. Παρακάτω βλέπουμε και μια βοηθητική γραφική απεικόνιση αυτού του πίνακα.



Εικόνα 6: Παράδειγμα εύρεσης του πίνακα acc

Για όλους τους πίνακες που χρησιμοποιούμε στην εργασία γίνεται δυναμική εκχώρηση μνήμης, ώστε να ελευθερώνουμε την μνήμη που καταλαμβάνουν όταν δεν τους χρειαζόμαστε πια. Όπως κάνουμε και για τους T_init και T_filled αφού δημιουργήσουμε τους πίνακες T και L [27], όπως βλέπουμε παρακάτω. Αυτή η διαδικασία είναι πολύ χρήσιμη γιατί θα κερδίσουμε αρκετό χώρο στην κύρια μνήμη για τις μετέπειτα διεργασίες που επιθυμούμε να πράξουμε.

```
//edw kanoume apeleutherwsh tous pinakes T_init kai T_fiiled afou pleon de
//mas xreiazontai afou exoume tous T kai L
for(i=0;i<G->upsos;i++){
    free(T_init[i]);
}
free(T_init);
free(T_filled);
```

3.4 Βοηθητικές συναρτήσεις

Για την υλοποίηση των διάφορων ερωτημάτων αλλά και του κύριου κώδικα χτισίματος του αλγορίθμου μας, χρειαζόμαστε κάποιες βοηθητικές συναρτήσεις. Σε αυτές τις συναρτήσεις θα αναφερθούμε σε αυτή την παράγραφο.

3.4.1 Συνάρτηση υπολογισμού της τάξης

Σε αυτή τη συνάρτηση με τη βοήθεια του πίνακα acc, που έχουμε αναφέρει παραπάνω τη λειτουργία του, θα υπολογίσουμε την τάξη του πίνακα T. Ο υπολογισμός της τάξης είναι απαραίτητος για την υλοποίηση των διάφορων ερωτημάτων που θα δούμε στις επόμενες παραγράφους. Αυτή η συνάρτηση σαν παραμέτρους έχει μια μεταβλητή G τύπου compressed graph (sturct που περιέχει όλες τις απαραίτητες μεταβλητές) και μια ακέραια μεταβλητή position, όπου μας δείχνει τη θέση του στοιχείου στο οποίο βρισκόμαστε στον πίνακα T. Τελικά αυτή η συνάρτηση επιστρέφει το πλήθος των μονάδων μέχρι τη θέση position.

Αν η μεταβλητή position πάρει την τιμή -1, τότε η τάξη είναι 0, αφού βρισκόμαστε πριν το πρώτο στοιχείο του πίνακα T. Αυτό συμβαίνει γιατί αυτή η μεταβλητή αντιστοιχεί στη μεταβλητή z (αντίστοιχα μας δείχνει τη θέση που βρισκόμαστε στον πίνακα T), την οποία χρησιμοποιούμε σε όλες τις συναρτήσεις μας και πάντα σαν αρχική τιμή παίρνει το -1. Αν η μεταβλητή position έχει την τιμή 0 τότε βρισκόμαστε στο πρώτο στοιχείο του πίνακα T κι επιστρέφεται ακριβώς η τιμή που βρίσκεται σε αυτή τη θέση (0 ή 1), αφού θα είναι κι η αντίστοιχη τάξη στη θέση 0 του πίνακα T.

Για τη διευκόλυνση στον υπολογισμό της τάξης θα χρησιμοποιήσουμε και τον πίνακα acc, στον οποίο έχουμε αναφερθεί λεπτομερώς σε προηγούμενο κεφάλαιο. Αν η θέση (δηλαδή η μεταβλητή position) που βρισκόμαστε στον πίνακα T είναι ακέραιο πολλαπλάσιο του X (ανά X στοιχεία τοποθετούμε το πλήθος των μονάδων του T στον acc), τότε η μεταβλητή plithos_monadwn (είναι μια βοηθητική μεταβλητή που κρατάει το πλήθος των μονάδων) παίρνει ακριβώς την τιμή του acc που βρίσκεται στη θέση position/X (πάντα κρατάμε την ακέραια τιμή). Αν δεν είναι ακέραιο πολλαπλάσιο, τότε η μεταβλητή plithos_monadwn παίρνει την τιμή της αμέσως προηγούμενης τάξης που είναι αποθηκευμένη στον πίνακα acc και προσθέτουμε σε αυτήν το πλήθος των μονάδων που υπάρχουν μέχρι τη θέση position. Αν για παράδειγμα έχουμε τον πίνακα T της εικόνας 5 και θέλουμε να βρούμε την τάξη στη θέση, τότε αρκεί να κάνουμε την εξής πράξη : acc[0]+2=2+2=4. Άρα η τάξη στη θέση 6 θα είναι ίση με 4. Υπάρχει και το ενδεχόμενο να είμαστε στις πρώτες θέσεις, δηλαδή πριν ακόμη ορίσουμε την πρώτη τιμή στον πίνακα acc, σε αυτή την περίπτωση αρκεί να υπολογίσουμε το πλήθος των μονάδων μέχρι τη θέση που βρισκόμαστε. Παρακάτω βλέπουμε και τον κώδικα που φτιάξαμε για τον υπολογισμό της τάξης.

```
/*sunarthsh pou upologizei thn taksi se enan pinaka (tou T)*/
int rank(compressed_graph_T * G, int position){/*opou position einai h thesi
     tou stoixeiou mas ston pinaka T*/
    int plithos;/* plithos einai o metavliti pou tha mas deixnei posoi einai
    assoi apo th thesi pou exoume vrei thn kontinoteri taksi mexri to z
    (pou z einai h thesi pou vriskomaste ston T ),
    thn taksi thn apothikeoume ston pinaka acc (acc:pinakas pou krataei tin
    taksi ana x stoixeia tou pinaka T) mexri to zhtoumeno z */
    int i;
    int plithos_monadwn;/*opou plithos_monadwn einai  metavliti pou apothikeume
     to plithos twn monadwn mexri
    mia sugkekrimeni thesi*/

    plithos=0;/*arxikopoihsh ths metavlitis plithos*/

    if(position==-1){/*an positios=-1 tote h taksi einai 0,ara plithos_monadwn=0,
            opou position einai trexousa thesi mas ston T*/
        return 0;
    }
    else if (position ==0){/*an eimaste sthn prwth thesi tou T tote theloume na
        mas epistrefete to stoixeio
            pou exei h prwth thesi 0 h 1*/
        return H(G->T,0);
    }
    else if((position+1)%G->X==0){/*edw an h thesi tou stoixeiou mas einai z diladi
         pol/sio tou X tote h taksi mexri ekeino to stoixeio tha einai akrivws oso
         h taksi pou einai apothikeumeni sth thesi z/X tou pinaka acc
                    (vazoume kai -1 ston tupo giati ksekiname thn arithmisi twn
                     stoixeiwn tou acc apo to 0)*/
         plithos_monadwn=G->acc[position/G->X];
    }
    else {/*an to z den einai pol/sio tou X tote apla pame kai vriskoume apo ton
        pinaka acc thn ameswws prohgoumenh
            taksi kai prosthetoume tous assous apo ekeinh thn thesi mexri to z*/
        for(i=position-(position%G->X);i<=position;i++){
            if(H(G->T,i)){/*an einai 1 to stoixeio mas prosthetoume mia monada
                 sto plithos*/
                    plithos++;
            }
            if (position+1 > G->X){/*an eimaste stis theseis meta tis X prwtes
                tote arkei na prosthesoume to acc sth thesi
                    (position/X)-1 kai to plithos twn asswn mexri th sygkekrimenh
                     thesi pou zhtame*/
                plithos_monadwn=G->acc[(position/G->X)-1]+plithos;
            }
            else{/*an eimaste se kapoia apo tis prwtes X-i(i=0...X-1) theseis tote
                oi assoi mas einai osoi einai ki h metavliti

                    plithos,afou sth thesi 0 den exoume stoixeia ston acc.
                    O acc to prwto stoixeio to pairnei otan ftasoume
                    sthn thesi X*/
                plithos_monadwn=plithos;
            }
        }
    }
    return(plithos_monadwn);
}
```

Όπως έχουμε εξηγήσει στον δεύτερο κεφάλαιο για να εφαρμοστεί ο κώδικάς, τον οποίο πραγματεύεται η εργασία, πρέπει οι διαστάσεις του πίνακα γειτνιάσεως να είναι κάποια δύναμη του k, υπάρχει όμως το ενδεχόμενο να μην είναι δύναμη του k. Για αυτόν τον λόγο απαιτείται ένας έλεγχος για τον αν οι διαστάσεις του πίνακα είναι ή όχι δύναμη του k [28,29]. Αυτόν τον έλεγχο τον κάνουμε

με τη συνάρτηση Power, η οποία επιστρέφει την τιμή 1 αν οι διαστάσεις είναι δύναμη του k ή 0 αν δεν είναι.

```
/*edw ginetai elegxos an enas arithmos n einai dunami tou k. H sunasthrh tha
gurisei 1 an to n einai dunami tou K alliws tha gurisei 0*/
int power(int K,int n){
    while (n != 1)
      {
        if (n%K != 0)
          return 0;
        n = n/2;
      }
      return 1;
}
```

Τελικά, αν οι διαστάσεις του πίνακα μας δεν είναι κάποια δύναμη του k, τότε επεκτείνουμε τον πίνακά μας. Συνεπώς, το μέγεθος του πίνακα μας θα αλλάξει, για τον υπολογισμό των νέων διαστάσεων του πίνακα μας χρησιμοποιούμε τη συνάρτηση extension.

```
/*me aythn thn sunarthsh upologizoume to megethos tou epektamenoy pinaka*/
int extension(int K,int N){
    int counter;
    int epektasi,N_new;

    counter=0;

    epektasi=power(K,N);/*xrhsh ths parapanw sunarthshs (power) gia thn metavliti
     megethos*/

     if (epektasi==1){
        N_new=N;
     }
     else{
        while (N>= 1){
            N= N/K;
            counter++;
         }

         N_new=pow(K,counter);
     }

    return (N_new);
}
```

3.5 Queries

Σε αυτή την παράγραφο θα παρουσιαστούν τα διάφορα ερωτήματα (queries) που έχουν τεθεί με σκοπό να ελεγχθεί η αποτελεσματικότητα της συμπίεσης που πραγματοποιήσαμε.

3.5.1 Έλεγχος σύνδεσης μεταξύ δύο κόμβων

Έχουμε δημιουργήσει μια συνάρτηση η οποία μπορεί να ελέγξει αν υπάρχει δεσμός (ακμή) μεταξύ δύο κόμβων. Παρακάτω θα την αναλύσουμε αλλά και θα παρουσιάσουμε τον κώδικα που εφαρμόσαμε για την υλοποίησή της. Στις περισσότερες αναπαραστάσεις συμπιεσμένων γραφημάτων για να ελεγχθεί αν δύο κόμβοι, ας τους ονομάσουμε p και q, συνδέονται μεταξύ τους πρέπει να εξαχθούν οι successors του κόμβου p και να ελέγξουμε αν σε αυτούς περιλαμβάνεται ο κόμβος q. Η συνάρτηση που αναπτύξαμε ελέγχει αν υπάρχει κάποιος σύνδεσμος (στον πίνακα γειτνιάσεως αν υπάρχει μονάδα σημαίνει πως δύο κόμβοι έχουν δεσμό μεταξύ τους, αν υπάρχει μηδενικό σημαίνει πως δεν έχουν κάποιο δεσμό) χωρίς να εξάγουμε όλους τους successors μιας σελίδας. Εμείς μπορούμε να απαντήσουμε στο ερώτημα αν υπάρχει δεσμός μεταξύ δύο κόμβων σε χρόνο O($\log_k n$) (όπου k είναι η τιμή που εισάγει ο χρήστης και n το μέγεθος του πίνακα γειτνιάσεως).

Κάθε παιδί, που βρίσκεται στους εσωτερικούς κόμβους (δηλαδή όχι στα φύλλα του δένδρου), αναπαριστά έναν ολόκληρο υποπίνακα του αρχικού μας πίνακα γειτνιάσεως (A). Εμείς ξεκινώντας από τη ρίζα φθίνουμε προς τον κόμβο του παιδιού που αναπαριστά το κελί του πίνακα γειτνιάσεως που μας ενδιαφέρει, δηλαδή πάμε στο παιδί που αναπαριστά τον υποπίνακα, που περιέχει το στοιχείο Apq (δηλαδή το κελί του πίνακα γειτνιάσεως που διασταυρώνονται οι κόμβοι p και q).

Όπως έχουμε ορίσει παραπάνω το ύψος του δένδρου το βρίσκουμε από τον εξής τύπο $h = \lceil \log_k n \rceil$. Οι κόμβοι στο επίπεδο l του δένδρου μας αναπαριστούν τους υποπίνακες μεγέθους k^{h-l} κι αυτοί διασπώνται σε υποπίνακες μεγέθους k^{h-l-1}. Για παράδειγμα η ρίζα (l=0) αναπαριστά ολόκληρο τον πίνακα μεγέθους $k^h=n'$ (όπου n' είναι το μέγεθος του επεκτεταμένου πίνακα).

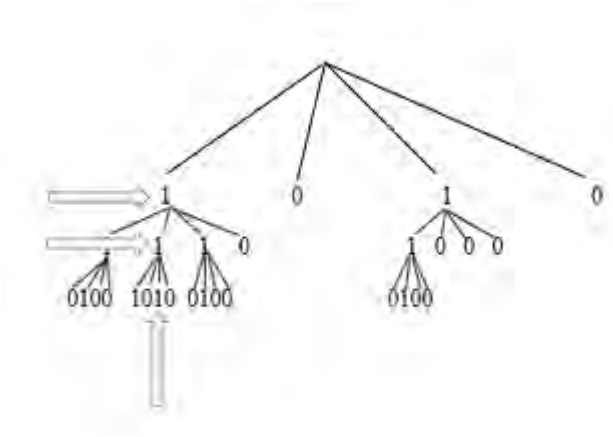
Ας ονομάσουμε p_l τη σχετική θέση της σειράς του κόμβου p στο επίπεδο l και q_l τη σχετική θέση της στήλης του κόμβου q στο επίπεδο l. Το κελί (p_l , q_l) ενός πίνακα του επιπέδου l ανήκει σε έναν υποπίνακα, δηλαδή στη σειρά $\lfloor p_l/k^{h-l-1} \rfloor$ και στη στήλη $\lfloor q_l/k^{h-l-1} \rfloor$. Αυτός ο υποπίνακας αντιστοιχεί στο παιδί $k*\lfloor p_l/k^{h-l-1} \rfloor+\lfloor q_l/k^{h-l-1} \rfloor$, που είναι το παιδί του κόμβου ο οποίος βρίσκεται στο επίπεδο l. Η σχετική θέση της σειράς για τον κόμβο p, σε αυτόν τον κόμβο παιδί, είναι $p_{l+1}=p_l \bmod k^{h-l-1}$. Ενώ η σχετική θέση της στήλης για τον κόμβο q είναι $q_{l+1}=q_l \bmod k^{h-l-1}$.

Δηλαδή η διαδικασία που ακολουθούμε είναι η εξής: ξεκινάμε από τη ρίζα που βρίσκεται στο επίπεδο l=0, όπου p_o=p και q_o=q και σε κάθε επίπεδο φθίνουμε προς το παιδί που βρίσκεται στη θέση $k*\lfloor p/k^{h-l-1} \rfloor+\lfloor q/k^{h-l-1} \rfloor$, αν στο παιδί που βρισκόμαστε η τιμή που περιέχεται είναι μονάδα, τότε υπολογίζουμε τη σχετική θέση του κελιού (p,q) στον αντίστοιχο υποπίνακα. Όταν φτάσουμε στο

τελευταίο επίπεδο του δένδρου μας κι υπάρχει μονάδα στο κελί (p,q) τότε υπάρχει σύνδεσμος (ακμή) μεταξύ των κόμβων μας, αλλιώς, δηλαδή η τιμή να είναι 0, δεν υπάρχει κάποιος σύνδεσμος. Γενικά σε αυτή τη συνάρτηση με αναδρομικές κλήσεις της φτάνουμε στο τελευταίο επίπεδο του δένδρου μας και βλέπουμε αν υπάρχει άσσος ή μηδενικό δηλαδή αν υπάρχει ακμή μεταξύ δύο κόμβων ή όχι αντίστοιχα.

Θα παρουσιάσουμε ένα παράδειγμα για την καλύτερη κατανόηση της λειτουργίας της συγκεκριμένης συνάρτησης. Ας πούμε πως θέλουμε να ελέγξουμε αν υπάρχει σύνδεσμος (ακμή) μεταξύ των κόμβων 2 και 3, δηλαδή πρέπει να ελέγξουμε αν στο κελί (2,3) του πίνακα υπάρχει μηδενικό ή άσσος. Ξεκινάμε από την ρίζα του δένδρου (για k=2) και φθίνουμε στο πρώτο από αριστερά παιδί της, αφού η μονάδα που βρίσκεται σε αυτή τη θέση αναπαριστά τον υποπίνακα που περιέχεται το κελί (2,3). Σε αυτόν τον κόμβο υπάρχει άσσος (αν βρίσκαμε μηδενικό, τότε θα σταματούσε η διαδικασία και σαν αποτέλεσμα δε θα υπήρχε σύνδεσμος μεταξύ των κόμβων) οπότε φθίνουμε στο δεύτερο από αριστερά παιδί του (αφού η μονάδα που βρίσκεται σε αυτή τη θέση αναπαριστά τον υποπίνακα που περιέχεται το κελί (2,3)), που βρίσκεται στο δεύτερο επίπεδο του δένδρου μας, και βλέπουμε πως κι αυτό το παιδί κόμβος περιέχει άσσο. Επομένως, πάμε στα παιδιά του δηλαδή στο αμέσως επόμενο επίπεδο, το τρίτο (φύλλα του δένδρου), και βλέπουμε τι ψηφίο περιέχει αυτός ο κόμβος παιδί στη θέση που μας ενδιαφέρει (δηλαδή τη (2,3)). Τελικά, βλέπουμε πως περιέχει μονάδα, οπότε καταλήγουμε στο συμπέρασμα πως μεταξύ των κόμβων 2 και 3 υπάρχει δεσμός.

Κάθε φορά αν ο κόμβος που μας ενδιαφέρει, δηλαδή αυτός που αναπαριστά τον υποπίνακα στον οποίο βρίσκεται το ψηφίο που θέλουμε να ελέγξουμε αν είναι μηδενικό ή άσσος, περιέχει μονάδα συνεχίζουμε στα παιδιά του και επαναλαμβάνουμε την ίδια διαδικασία μέχρι να φτάσουμε στα φύλλα του δένδρου ή μέχρι να συναντήσουμε μηδενικό. Την παραπάνω διαδικασία ακολουθούμε αν έχουμε συμπίεση μόνος τους άσσους. Παρακάτω βλέπουμε κι ένα σχεδιάγραμμα με την πορεία που ακολουθήσαμε για την εύρεση του ψηφίου στη θέση 2,3.



Εικόνα 7: Παράδειγμα εύρεσης συνδέσμου μεταξύ δύο κόμβων.

Τη συνάρτηση για την εύρεση συνδέσμου μεταξύ δύο κόμβων την ονομάσαμε check_link(G,n',p,q,z). Οι παράμετροι είναι οι εξής: μια μεταβλητή τύπου compressed graph, το μέγεθος του πίνακα γειτνιάσεως (επαυξημένο ή όχι), η σειρά που μας ενδιαφέρει στον εκάστοτε υποπίνακα, η στήλη που μας ενδιαφέρει στον εκάστοτε υποπίνακα και η τελευταία παράμετρος απεικονίζει τη θέση που είμαστε στον T ή L (η αρχική του τιμή είναι -1, αφού τα δένδρα μας δεν αναπαριστούν τη ρίζα). Επίσης, υποθέτουμε ότι η τάξη είναι ίση με 0 όταν είμαστε στη ρίζα, δηλαδή η τελευταία μεταβλητή είναι ίση με -1. Παρακάτω βλέπουμε την υλοποίηση της συνάρτησης check_link σε C.

```
/*me authn th sunarthsh elegxoume an dyo komboi sundeontai,to n einai to megethos
 tou pinaka ,ta p kai q einai oi komvoi
pou hteloume na doume an uparxei kapoio link metaksu tous kai to z einai h thesi
toy komvou mas ston T h ston L
(analoga se poio epipedo tou dendrou mas vriskomaste.O T einai o pinakas pou
 periexei sumpiesmena ola ta stoixeia tou pinaka
 geitniasews mexri kai to proteleutaio epipedo, enw o L o pinakas pou periexei
 ta stoixeia tou teleutaiou epipedou).
 H sunarthsh epistrefei thn timi 1 an uparxei sundesh metaksu duo komvwn alliws
  epistrefei thn timi 0*/
int check_link(compressed_graph_T  G, int n,int p,int q,int z){
    int y,i;
    int flag;/*einai mia metavliti (pairnei tis times 0 h 1) pou tha mas voithisei
     gia na kseroume an exoume K*K mhdenika
     sth seira  ,an flag==1 tote exoume K*K mhdenika sth seira alliws flag==0*/

/*an eimaste sto teleutaio epipedo (dhladh ston L) epistrefetai h timh tou pinaka
L(0 h 1 ),opote an h timi einai 1 uparxei link
alliws den uparxei*/
    if(z>=G.T_size){
        return(H(G.L,(z-G.T_size)));
    }

    else{
        if(z==-1 || H(G.T,z) ) { /*to z pairnei thn timi -1 otan eimaste sthn riza
                afou emeis ksekiname thn
               arithmisi twn level apo to  1*/
            y=rank(&G,z)*(G.K*G.K);/*to y einai h metavliti pou tha krataei kathe
        fora se poio paidi prepei na pame
            wste na ftasoume ston L gia na doume an uparxei 0 h 1 sth sthlh kai sth
             seira pou theloume*/

           flag = 1;/*dhladh me ton parakatw kwdika me to pou vrethoun K*K mhdenika
           termatizei ki epistrefei 1 giati o upopinakas tha einai olos me monades
           ara
           sigoura tha uparxei link metaksu twn komvwn*/
           if(y<G.T_size){
                for (i=y; i<y+G.K*G.K; i++){
                    if (H(G.T,i) == 1)
                        flag = 0;
                }
                if (flag)
                    return(1);
            }

            y=y+floor(p/(n/G.K))*G.K+floor(q/(n/G.K));
            check_link(G,n/G.K,p%(n/G.K),q%(n/G.K),y);/*spame se upopinakes ton arxiko
             mas pinaka
             wste na ftasoume ston upopinaka pou periexei th sthlh kai th seira pou
             mas endiaferei*/
        }
        else{
            return 0;
       }
    }
}
```

Αρχικά για τη συνάρτηση εύρεσης συνδέσμου μεταξύ δύο κόμβων (check_link) δημιουργήσαμε τον κώδικα έτσι ώστε να λειτουργεί μόνο για τη συμπίεση των μηδενικών. Στη συνέχεια προσθέσαμε ένα κομμάτι κώδικα ώστε να είναι αποδοτικός και στη συμπίεση των άσσων, το παρακάτω:

```
    flag = 1;/*dhladh me ton parakatw kwdika me to pou vrethoun K*K mhdenika
    termatizei ki epistrefei 1 giati o upopinakas tha einai olos me monades
    ara
    sigoura tha uparxei link metaksu twn komvwn*/
    if(y<G.T_size){
         for (i=y; i<y+G.K*G.K; i++){
             if (H(G.T,i) == 1)
                 flag = 0;
         }
         if (flag)
             return(1);
     }

     y=y+floor(p/(n/G.K))*G.K+floor(q/(n/G.K));
     check_link(G,n/G.K,p%(n/G.K),q%(n/G.K),y);/*spame se upopinakes ton arxiko
      mas pinaka
      wste na ftasoume ston upopinaka pou periexei th sthlh kai th seira pou
      mas endiaferei*/
}
```

Όπως είδαμε σε προηγούμενα κεφάλαια κάθε φορά που συναντάμε έναν υποπίνακα γεμάτο με άσσους, τον απεικονίζουμε στο δένδρο μας με μια μονάδα και για παιδιά του κόμβου, που περιέχει αυτή τη μονάδα, τοποθετούμε k*k μηδενικά. Παραπάνω βλέπουμε τον κώδικα που εφαρμόσαμε για τη συμπίεση των άσσων. Αρχικά θέτουμε μια βοηθητική μεταβλητή flag (τύπου int) ίση με 1, όταν αυτή η μεταβλητή είναι ίση με 1 σημαίνει πως ένας ολόκληρος υποπίνακας είναι γεμάτος με μονάδες. Αναγνωρίζουμε κάθε φορά πως ένας υποπίνακας είναι γεμάτος με άσσους αν τα παιδιά ενός κόμβου είναι μηδενικά, δηλαδή κάνουμε αναζητήσεις στον πίνακα T ανά k*k στοιχεία, αν είναι όλα μηδενικά σημαίνει πως ο κόμβος γονιός τους είναι μονάδα κι αναπαριστά ένα υποπίνακα με μονάδες. Αν η μεταβλητή flag είναι ίση με 1, τότε η μια κλήση της συνάρτησής μας τερματίζει κι επιστρέφεται η τιμή 1.

3.5.2 Εύρεση successors και predecessors

Σε αυτήν την υποπαράγραφο θα παρουσιάσουμε τη διαδικασία εύρεσης των successors και predecessors ενός κόμβου p(q). Για να βρούμε τους successors (predecessors) ενός κόμβου αρκεί να εξάγουμε από τη σειρά p (στήλη q) τη θέση των στηλών (γραμμών) που περιέχουν άσσους. Και σε αυτή τη συνάρτηση χρησιμοποιούμε μια από πάνω προς τα κάτω αναζήτηση του δένδρου (top-down tree traversal), αλλά σε αντίθεση με την προηγούμενη συνάρτηση που επιλέγουμε κάθε φορά ένα παιδί, σε αυτόν τον αλγόριθμο κάθε φορά επιλέγονται k παιδιά από τα k*k παιδιά ενός κόμβου.

Ας ονομάσουμε p_l τη σχετική θέση της σειράς (q_l τη σχετική θέση στήλης), που μας ενδιαφέρει, στο επίπεδο l. Η σειρά p_l του υποπίνακα του επιπέδου l αντιστοιχεί στο παιδί k*$\lfloor p_l/k^{h-l-1} \rfloor$+j, 0≤ j <k. Παρομοίως, η στήλη q_l του υποπίνακα του επιπέδου l αντιστοιχεί στο παιδί j*k +$\lfloor q_l/k^{h-l-1} \rfloor$+j, 0≤ j <k.

Θα παρουσιαστεί ένα παράδειγμα για την καλύτερη κατανόηση της διαδικασίας εύρεσης των successors (την αντίστοιχη διαδικασία ακολουθούμε για την εύρεση των predecessors). Ας υποθέσουμε ότι θέλουμε να εξάγουμε τους successors του κόμβου 3 από την εικόνα 2 για k=2. Βλέπουμε πως ο κόμβος 3 αναπαρίσταται στη γραμμή p_0 = 3 όταν είμαστε στο επίπεδο l = 0, αφού σε αυτό το επίπεδο αντιστοιχεί ολόκληρος ο πίνακας γειτνιάσεως. Όταν l = 1, τότε η σχετική θέση της σειράς 3 μέσα στους υποπίνακες μεγέθους 4x4 (που βρίσκονται στο πάνω μέρος του πίνακά μας) είναι p_1=3mod4=3. Τελικά, όταν είμαστε στο επίπεδο l = 2 η σχετική θέση της σειράς μας μέσα στους υποπίνακες μεγέθους 2x2 είναι p_2= 3mod2=1.

Παρακάτω θα δούμε τον κώδικα για την υλοποίηση αυτής της διαδικασίας, δηλαδή για την εύρεση των successors αλλά και των predecessors. Έχουμε λοιπόν δημιουργήσει δύο συναρτήσεις τις successors(G,n',p,q,z) και predecessors(G,n',q,p,z). Στην πρώτη συνάρτηση οι παράμετροι είναι οι εξής : το G που είναι μια μεταβλητή τύπου compressed graph, το n' που είναι το μέγεθος του, επεκτεταμένου ή μη, πίνακα, το p που είναι η γραμμή από την οποία θέλουμε να εξάγουμε τους successors, το q είναι η μεταβλητή που αποθηκεύουμε κάθε φορά τη στήλη που υπάρχει άσσος και η τελευταία μεταβλητή είναι η θέση που βρισκόμαστε στον πίνακα T. Στη δεύτερη συνάρτηση οι παράμετροι είναι οι εξής : το G που είναι μια μεταβλητή τύπου compressed graph, το n' που είναι το μέγεθος του, επεκτεταμένου ή μη, πίνακα, το q που είναι η στήλη από την οποία θέλουμε να εξάγουμε τους predecessors, το p είναι η μεταβλητή που αποθηκεύουμε κάθε φορά τη γραμμή που υπάρχει άσσος και η τελευταία μεταβλητή είναι η θέση που βρισκόμαστε στον πίνακα T. Οι αρχικές τιμές των παραπάνω συναρτήσεων είναι οι εξής : successors(G,n',p,0,-1) και predecessors(G,n',q,0,-1).

Όπως έχουμε αναφέρει παραπάνω με τη συνάρτηση successors εξάγουμε τη θέση των στηλών, κάθε σειράς, που περιέχεται άσσος και στη συνάρτηση predecessors εξάγουμε τη θέση των σειρών, κάθε στήλης, που περιέχεται άσσος. Για παράδειγμα οι successors για τη γραμμή 3 (εικόνα 2 για k=2) είναι μόνο το {2}, αφού εκεί μόνο υπάρχει άσσος στην γραμμή 3. Επίσης, οι predecessors για τη στήλη 3 είναι οι {1,2}. Παρακάτω βλέπουμε και την υλοποίηση σε C των παραπάνω συναρτήσεων.

```
/*me auth thn sunarthsh vriskoume tous successors mia selidas p,successors ennooyme
 to plithos twn istoselidwn pou deixnontai
apo kapoies alles.Pairnoume kathe seira apo ton pinaka mas kai vlepoume se poies
sthles uparxoun assoi.Telika
h sunarthsh mas epistrefei to noumero ths sthlhs pou uparxei assos ki auto ginetai
 gia kathe grammi.Epeidi ayth h synarthsh
epistrefei to noumero olwn twn sthlwn ,apo kathe grammi, tha topothethsoume auta ta
noumera se mia lista ,opou
kathe komvos tha periexei to noumero ths sthlhs pou vrisketai assos apo sugkerimenh
 grammi*/
void successors(compressed_graph_T * G,int n,int p,int q,int z){
    int y,i,flag,j;
    node_T *new_node;

    if(z>=G->T_size){
        if(H(G->L,(z-G->T_size))){/*an eimaste sto teleutaio epipedo tote tha mas
                epistrefetai h timi tou q*/
            new_node=(node_T*)malloc(sizeof(node_T));
            if(new_node==NULL){
                printf("Error in successors");
                exit(1);
            }
            new_node->page=q;
            new node->next=G->s head;

            G->s_head=new_node;
        }
    }
    else{
        if(z==-1 || H(G->T,z) ){
           y=rank(G,z)*(G->K*G->K);

           flag = 1;
           /*dhladh auto o kwdikas me to pou vrethoun K*K mhdenika
           (auto sumvainei giati kathe fora pou sunantame ena -1 ston T emeis
            antikathisoume me monada to -1 kai vazoume gia paidia tou K*K mhdenika)
           termatizei ki epistrefontai oi sthles pou periexoun assous,pou tha einai
            sunexomenes (stis sugkekrimenes grammes pou theloume ki uparxoun assoi)
            afou o upopinakas mas einai gematos me assous*/
           if(y<G->T_size){/*an eimaste sta oria tou pinaka T*/
                for (i=y; i<y+G->K*G->K; i++){
                    if (H(G->T,i) == 1)
                        flag = 0;
                }
                if (flag){/*tote olos o upopinakas  me riza T[z] tha periexei assous.
                        Se auth thn periptwsh enas upopinakas
                        periexei mono assous opote apo th thesi pou ksekinaei o upopinakas
                        tha exoume se sunexomenes theseis
                        successors*/
                   for(j=0;j<G->K*G->K;j++){
                       new_node=(node_T*)malloc(sizeof(node_T));
                       if(new_node==NULL){
                           printf("Error in successors");
                           exit(1);
                       }
                       new_node->page=q+j;
                       new_node->next=G->s_head;
                       G->s_head=new_node;
                   }
                }

               else{/*alliws sunexizetai h anadromi,giati de vrikame upodendro
                    gemato me assous*/
                    y=rank(G,z)*(G->K*G->K)+G->K*floor(p/(n/G->K));
                    for(i=0;i<G->K;i++){
                        successors(G,n/G->K,p%(n/G->K),q+(n/G->K)*i,y+i);
                    }
               }
           }
    }
```

```
            else{
                y=rank(G,z)*(G->K*G->K)+G->K*floor(p/(n/G->K));
                for(i=0;i<G->K;i++){
                    successors(G,n/G->K,p%(n/G->K),q+(n/G->K)*i,y+i);
                }
            }
        }
    }
}



/*me auth thn sunarthsh vriskoume tous predecessors mia selidas p,predecessors
 ennooyme to plithos twn istoselidwn pou deixnoun
se kapoies alles selides.Pairnoume kathe sthlh apo ton pinaka mas kai vlepoume
se poies grammes uparxoun assoi ki h sunarthsh
epistrefei autes tis grammes.Telika h sunarthsh mas epistrefei to noumero ths
 grammis pou uparxei assos ki auto
ginetai gia kathe sthlh (pou emeis theloume).Epeidi ayth h synarthsh epistrefei
to noumero olwn twn grammwn
,apo kathe sthlh, tha topothethsoume auta ta noumera se mia lista ,opou
kathe komvos tha periexei to noumero ths grammis pou vrisketai assos apo sugkerimenh
 sthlh*/
void predecessors(compressed_graph_T  * G,int n,int q,int p,int z){
    int y,i,flag,j;
    node_T *n_node;


    if(z>=G->T_size){
        if(H(G->L,(z-G->T_size))){/*an eimaste sto teleutaio epipedo tote tha mas
                epistrefetai h timi tou p*/
            n_node=(node_T*)malloc(sizeof(node_T));
            if(n_node==NULL){
                printf("Error predecessors");
                exit(1);
            }

            n_node->page=p;
            n_node->next=G->p_head;
            G->p_head=n_node;
        }
    }
    else{
        if(z==-1 || H(G->T,z) ){
           y=rank(G,z)*(G->K*G->K);
           flag = 1;
           /*dhladh auto o kwdikas me to pou vrethoun K*K mhdenika
           (auto sumvainei giati kathe fora pou sunantame ena -1 ston T emeis
             antikathisoume me monada to -1 kai vazoume gia paidia tou K*K mhdenika)
           termatizei ki epistrefontai oi grammes pou periexoun assous,pou tha einai
           sunexomenes (stis sugkekrimenes sthles pou theloume ki uparxoun assoi)
           afou o upopinakas mas einai gematos me assous*/
           if(y<G->T_size){
                for (i=y; i<y+G->K*G->K; i++){
                    if (H(G->T,i) == 1)
                        flag = 0;
                }
                if (flag){/*tote olos o upopinakas  me riza T[z] tha periexei assous.
                        Se auth thn periptwsh enas upopinakas
                        periexei mono assous opote apo ti thesi pou ksekinaei o
```

```
                    upopinakas tha exoume se sunexomenes theseis
                    predecessors*/
                for(j=0;j<G->K*G->K;j++){
                    n_node=(node_T*)malloc(sizeof(node_T));
                    if(n_node==NULL){
                        printf("Error predecessors");
                        exit(1);
                    }
                    n_node->page=p+j;
                    n_node->next=G->p_head;
                    G->p_head=n_node;
                }
            }

            else{/*an den einai o upopinakas gematos assous sunexizei h anadromi*/
                y=rank(G,z)*(G->K*G->K)+floor(q/(n/G->K));
                for(i=0;i<G->K;i++){
                    predecessors(G,n/G->K,q%(n/G->K),p+(n/G->K)*i,y+i*G->K);
                }
            }
        }

        else{
            y=rank(G,z)*(G->K*G->K)+floor(q/(n/G->K));
            for(i=0;i<G->K;i++){
                predecessors(G,n/G->K,q%(n/G->K),p+(n/G->K)*i,y+i*G->K);
            }
        }
    }
}
```

Εμείς για κάθε μία από τις παραπάνω συναρτήσεις δημιουργούμε μια απλά διασυνδεδεμένη χωρίς τερματικό λίστα ώστε να αποθηκεύουμε τα αποτελέσματα, δηλαδή τις στήλες που περιέχουν άσσους για τους successors και τις γραμμές για τους predecessors. Πάντα τα στοιχεία μας αποθηκεύονται στους κόμβους της λίστας ,που δημιουργούμε, με τη σειρά που τα εξάγουμε. Παρακάτω βλέπουμε και το stuct που περιέχει τις μεταβλητές που είναι απαραίτητες για τη δημιουργία μιας λίστας. Όλα τα structs τα έχουμε δημιουργήσει για να αποφύγουμε τη χρήση global μεταβλητών.

```
struct node {
    struct node * next;
    int page;
    int row;
    int column;
};

typedef struct node node_T;
```

Τις συναρτήσεις successors και predecessors για το ίδιο δένδρο μπορούμε να τις καλέσουμε επανελειμμένα για διάφορες τιμές (δηλαδή για διάφορες σειρές ή στήλες αντίστοιχα), αυτό δε θα ήταν δυνατό αν δεν αποδεσμεύσαμε τη μνήμη που καταλαμβάνει κάθε λίστα μετά από κάθε κλήση των συναρτήσεών μας. Οπότε δημιουργήσαμε τη συνάρτηση clear_list ώστε να την καλούμε πριν από κάθε κλήση των συναρτήσεων μας και να είμαστε σίγουροι πως οποιαδήποτε λίστα θα χωρέσει στη μνήμη του υπολογιστή μας.

```
/*me auth th sunarthsh katharizoume tis listes mas ,wste na mporoume na enthesoume
 oses fores theloume stoixeia se autes.*/
node_T *clear_list(node_T *p) {
   node_T *temp;
   if (p!=NULL) {
       temp = p->next;
       free(p);
       p=clear_list(temp);
   }
   return(p);
}
```

Όπως και στη συνάρτηση check_link, έτσι κι σε αυτές τις συναρτήσεις ο αρχικός τους κώδικας ήταν έτσι χτισμένος ώστε να εξάγονται οι successors κι οι predecessors εφόσον έχουμε συμπίεση των μηδενικών μόνο. Στη συνέχεια προσθέσαμε τα παρακάτω κομμάτια κώδικα ώστε οι συναρτήσεις μας να είναι αποτελεσματικές και μετά τη συμπίεση των άσσων.

```
flag = 1;
/*dhladh auto o kwdikas me to pou vrethoun K*K mhdenika
(auto sumvainei giati kathe fora pou sunantame ena -1 ston T emeis
 antikathisoume me monada to -1 kai vazoume gia paidia tou K*K mhdenika)
termatizei ki epistrefontai oi sthles pou periexoun assous,pou tha einai
 sunexomenes (stis sugkekrimenes grammes pou theloume ki uparxoun assoi)
 afou o upopinakas mas einai gematos me assous*/
if(y<G->T_size){/*an eimaste sta oria tou pinaka T*/
     for (i=y; i<y+G->K*G->K; i++){
         if (H(G->T,i) == 1)
             flag = 0;
     }
     if (flag){/*tote olos o upopinakas  me riza T[z] tha periexei assous.
             Se auth thn periptwsh enas upopinakas
             periexei mono assous opote apo th thesi pou ksekinaei o upopinakas
             tha exoume se sunexomenes theseis
             successors*/
         for(j=0;j<G->K*G->K;j++){
             new_node=(node_T*)malloc(sizeof(node_T));
             if(new_node==NULL){
                 printf("Error in successors");
                 exit(1);
             }

             new_node->page=q+j;
             new_node->next=G->s_head;
             G->s_head=new_node;
         }
     }
```

Ο κώδικας που βλέπουμε στο πάνω μέρος αφορά την προσθήκη που κάναμε στη συνάρτηση successors ώστε η συνάρτησή μας να είναι αποτελεσματική και μετά τη συμπίεση των άσσων, αφού αρχικά την είχαμε χτίσει με τέτοιον τρόπο ώστε να είναι αποτελεσματική μόνο για τη συμπίεση των μηδενικών. Όπως βλέπουμε έχουμε μια μεταβλητή flag η οποία είναι ίση με ένα όταν ένας υποπίνακας είναι γεμάτος με μονάδες. Θα διατρέχουμε τον πίνακα T ανά kxk στοιχεία και κάθε φορά που θα συναντάμε kxk μηδενικά, θα σημαίνει πως ο γονιός τους θα είναι αναπαριστά έναν υποπίνακα που θα είναι γεμάτος με μονάδες. Οπότε εμείς θα πρέπει να προσθέσουμε σε κάθε κελί της λίστα μας τη θέση

των συνεχόμενων στηλών που συναντάμε μονάδες. Αν για παράδειγμα θέλουμε να εξάγουμε τους successors μιας σειράς της οποίας ξέρουμε πως τα τελευταία 4 στοιχεία ανήκουν σε ένα υποπίνακα που περιέχει μόνο μονάδες (επομένως στον πίνακα T θα απεικονίζονται με μία μόνο μονάδα, λόγω συμπίεσης των άσσων). Τότε αρκεί να βρούμε από ποια θέση, στη συγκεκριμένη σειρά, βρίσκεται το πρώτο στοιχείο του υποπίνακά μας (αυτός που είναι γεμάτος με άσσους) και να εξάγουμε τους successors, που θα βρίσκονται σε συνεχόμενες θέσεις. Για παράδειγμα αν το πρώτο στοιχείο του υποπίνακα (4x4) που είναι γεμάτος με μηδενικά βρίσκεται στη στήλη 3, οι υπόλοιποι successors θα είναι στις στήλες 4,5,6,7.

```
flag = 1;
/*dhladh auto o kwdikas me to pou vrethoun K*K mhdenika
(auto sumvainei giati kathe fora pou sunantame ena -1 ston T emeis
  antikathisoume me monada to -1 kai vazoume gia paidia tou K*K mhdenika)
termatizei ki epistrefontai oi grammes pou periexoun assous,pou tha einai
sunexomenes (stis sugkekrimenes sthles pou theloume ki uparxoun assoi)
afou o upopinakas mas einai gematos me assous*/
if(y<G->T_size){
     for (i=y; i<y+G->K*G->K; i++){
         if (H(G->T,i) == 1)
             flag = 0;
     }
     if (flag){/*tote olos o upopinakas  me riza T[z] tha periexei assous.
               Se auth thn periptwsh enas upopinakas
               periexei mono assous opote apo ti thesi pou ksekinaei o
               upopinakas tha exoume se sunexomenes theseis
               predecessors*/
         for(j=0;j<G->K*G->K;j++){
             n_node=(node_T*)malloc(sizeof(node_T));
             if(n_node==NULL){
                 printf("Error predecessors");
                 exit(1);
             }

             n_node->page=p+j;
             n_node->next=G->p_head;
             G->p_head=n_node;
         }
     }
```

Ο κώδικας που βλέπουμε στο πάνω μέρος αφορά την προσθήκη που κάναμε στη συνάρτηση predecessors για να λειτουργεί αποτελεσματικά η συνάρτηση και μετά τη συμπίεση των άσσων, αφού τη συγκεκριμένη συνάρτηση αρχικά την είχαμε στήσει έτσι ώστε να λειτουργεί μόνο για την συμπίεση των μηδενικών. Όπως βλέπουμε έχουμε μια μεταβλητή flag η οποία είναι ίση με ένα όταν ένας υποπίνακας είναι γεμάτος με μονάδες. Θα διατρέχουμε τον πίνακα T ανά kxk στοιχεία και κάθε φορά που θα συναντάμε kxk μηδενικά, θα σημαίνει πως ο γονιός τους θα είναι αναπαριστά έναν υποπίνακα που θα είναι γεμάτος με μονάδες. Οπότε εμείς θα πρέπει να προσθέσουμε σε κάθε της λίστας μας τη θέση των συνεχόμενων γραμμών που συναντάμε μονάδες. Αν για παράδειγμα θέλουμε να εξάγουμε τους predecessors μιας στήλης της οποίας ξέρουμε πως τα τελευταία 4 στοιχεία ανήκουν σε ένα υποπίνακα που περιέχει μόνο μονάδες (επομένως στον πίνακα T θα απεικονίζονται με μία μόνο μονάδα, λόγω συμπίεσης των άσσων). Τότε αρκεί να βρούμε από ποια θέση, στη συγκεκριμένη στήλη,

βρίσκεται το πρώτο στοιχείο του υποπίνακά μας (αυτός που είναι γεμάτος με άσσους) και να εξάγουμε τους predecessors, που θα βρίσκονται σε συνεχόμενες θέσεις. Για παράδειγμα αν το πρώτο στοιχείο του υποπίνακα (4x4), που είναι γεμάτος με μηδενικά, βρίσκεται στη γραμμή 3 οι υπόλοιποι predecessors θα είναι στις γραμμές 4,5,6,7.

3.5.3 Range

Αυτή η συνάρτηση μοιάζει αρκετά με αυτές που παρουσιάσαμε στην προηγούμενη παράγραφο, μόνο που δεν εισάγουμε όλα τα παιδιά $0 \leq j < k$ μιας γραμμής (ή στήλης), αλλά μόνο αυτά που πληρούν την ανίσωση $\lfloor q_1/k^{h-l-1} \rfloor \leq j < \lfloor q_2/k^{h-l-1} \rfloor$ ($\lfloor p_1/k^{h-l-1} \rfloor \leq j < \lfloor p_2/k^{h-l-1} \rfloor$) και την ονομάσαμε range. Με αυτή τη συνάρτηση εξάγουμε τις στήλες αλλά και τις σειρές που υπάρχουν άσσοι μεταξύ συγκεκριμένων ορίων στον πίνακα γειτνιάσεως ((p_1, p_2) και (q_1, q_2)). Με τη συνάρτηση range(G,n', $p_1, p_2, q_1, q_2, d_p, d_q, z$) μπορούμε να λύσουμε κι όλα τα προηγούμενα ερωτήματα που θέσαμε αρκεί να κάνουμε κάποιες μικρές αλλαγές. Δηλαδή για να εξάγουμε τους successors του p αρκεί να θέσουμε όπου $p_1 = p_2 = p$, $q_1 = 0$, $q_2 = n - 1$, για να εξάγουμε τους predecessors του q αρκεί να θέσουμε όπου $q_1 = q_2 = q$, $p_1 = 0, p_2 = n - 1$, για να ελέγξουμε αν υπάρχει κάποιος σύνδεσμος (ακμή) μεταξύ των κόμβων p και q αρκεί να θέσουμε όπου $p_1 = p_2 = p$, $q_1 = q_2 = q$, για να βρούμε τους successors ενός κόμβου p μέσα σε ένα εύρος $[q_1, q_2]$ θέτουμε $p_1 = p_2 = p$ και τέλος για να βρούμε τους predecessors ενός κόμβου q μέσα σε ένα εύρος $[p_1, p_2]$ θέτουμε $q_1 = q_2 = q$. Η συνάρτηση range έχει σαν αρχικές τιμές τις εξής : (G,n',p_1, p_2, q_1, q_2, 0,0,-1).

Με αυτή τη συνάρτηση πάμε σε κάθε υποπίνακα με όρια τα $[p_1, p_2]$ για τις γραμμές και $[q_1, q_2]$ για τις στήλες κι εξάγουμε τις σειρές ή τις στήλες που περιέχουν μονάδες. Εκτενέστερα το d_q (d_p) είναι η μεταβλητή που σε κάθε κλήση της συνάρτησής μας αποθηκεύεται η στήλη (γραμμή) του υποπίνακά μας (που έχει συγκεκριμένα όρια) για κάθε γραμμή (στήλη), δηλαδή για κάθε σειρά (στήλη) του υποπίνακά μας εξάγουμε τη θέση της στήλης (γραμμής) που περιέχει μονάδα και την αποθηκεύουμε σε μια απλά διασυνδεδεμένη χωρίς τερματικό λίστα. Τις διάφορες τιμές των d_p και d_q τις αποθηκεύουμε στην ίδια λίστα, δηλαδή σε κάθε κελί της περιέχονται δύο τιμές αυτές των (d_q, d_p). Θα αναφερθούμε σε ένα παράδειγμα για την καλύτερη κατανόηση της λειτουργίας της συγκεκριμένης συνάρτησης. Θα χρησιμοποιήσουμε τον πίνακα που παρουσιάσαμε στην εικόνα 2 για k = 2. Ας υποθέσουμε ότι τα όρια που μας δίνονται για τις γραμμές είναι τα $[p_1 = 1, p_2 = 3]$ και για τις στήλες $[q_1 = 1, q_2 = 3]$, οπότε είμαστε στον υποπίνακα :

0	1	1
0	0	1
0	1	0

Δηλαδή λίστα μας που θα περιέχει τις τιμές των (d_q, d_p) σε κάθε κελί, δηλαδή θα είναι :

- (2,1)
- (3,1)
- (3,2)
- (2,3)

Όπως και στις προηγούμενες συναρτήσεις που αναφερθήκαμε έτσι και σε αυτή αρχικά ο κώδικας που είχαμε γράψει ήταν έτσι χτισμένος ώστε να λειτουργεί η συνάρτησή μας για τη συμπίεση των μηδενικών μόνο, στη συνέχεια έπρεπε να την τροποποιήσουμε έτσι ώστε να λειτουργεί και μετά τη συμπίεση των άσσων. Παρακάτω βλέπουμε την υλοποίηση της συνάρτησης range.

```
/*me aythn th sunarthsh tha vroume ta links enos komvou me kapoious allous apo
ena diasthma [p1,p1] se ena diasthma [q1,q2].
Dhladi tha vroume se poies theseis uparxoun links metaksu twn disthmatwn [p1,p1]
 kai [q1,q2] kai tha apothkeusoume se mia lista
,pou tha exei kefali to head_dp_dq, tis sthles (dq,pame se kathe grammi tou
upopinaka mas tou diasthmatos mas kai ekasgoume tis
sthles) kai tis grammes (dp,pame se kathe sthlh tou upopinaka mas tou diasthmatos
mas kai ekasgoume tis grammes) pou uparxoun assoi
metaksu autwn twn diasthmatwn.Kai se auth thn synarthsh tha xrhsimopoihsoyme lista
 gia na topothethhsoume auta pou mas epistrefei
pou einai ta dp kai dq.Opou dp einai oi grammes pou periexoun assous sta diathmata
 [p1,p2] kai [q1,q1] kai to dq oi sthles pou
periexoun assous stoa diasthmata [p1,p2] kai [q1,q1].Diladi vriskw toy successors
 tou p sto [q1,q2] kai tous predecessorstou
q sto [p1,p2] */
void range(compressed_graph_T * G,int n,int p1,int p2,int q1,int q2,int dp,int dq,int z){
/*auth einai h domh pou periexontai ta dp kai dq kai tha zhtame apo thn sunarthsh mas na
mas epistrefontai (diladi
zhtame ta dp kai dq ki epeidi de mporoume na kanoume epistrofh duo timwn gia auto
xrhsimopioume struct).Auth h domh range_T
periexei mono tis times pou theloume na epistrepsoume*/
    node_T * new_node;
    int i,j,y,flag,k,l;

    int p_1,p_2,q_1,q_2;

    if(z>=G->T_size){/*an eimaste sto teleutaio epipedo*/
        if(H(G->L,(z-G->T_size))){
            new_node=(node_T*)malloc(sizeof(node_T));
            if(new_node==NULL){
                printf("Error in range");
                exit(1);
            }
            new_node->column=dq;
            new_node->row=dp;
            new_node->next=G->head_dp_dq;
            G->head_dp_dq=new_node;
        }
    }
    else{
        if(z==-1 || H(G->T,z) ){
           y=rank(G,z)*(G->K*G->K);

           flag=1;
           /*dhladh auto o kwdikas me to pou vrethoun K*K mhdenika
           (auto sumvainei giati kathe fora pou sunantame
           ena -1 ston T_init emeis antikathisoume me monada to -1 kai vazoume gia

            paidia tou K*K mhdenika)
           termatizei ki epistrefontai oi grammes ki oi sthles pou periexoun
           assous,pou tha einai sunexomenes
           afou o upopinakas mas einai gematos me assous*/
           if(y<G->T_size){
                 for (i=y; i<y+G->K*G->K; i++){
                    if (H(G->T,i) == 1)
                        flag = 0;
                 }
                 if (flag){/*tote olos o upopinakas  me riza T[z] tha periexei assous.
                        Se auth thn periptwsh enas upopinakas
                        periexei mono assous opote tha apo th thesi pou ksekinaei o
                        upopinakas tha exoume se sunexomenes theseis
                        predecessors kai successors*/
                        range_all_one(G,n,p1,p2,q1,q2,dp,dq,z);
                 }
                 else{/*se periptwsh pou o upopinakas mas den einai gematos me assous sunexizetai h anadromi*/
                    for(i=floor(p1/(n/G->K));i<=floor(p2/(n/G->K));i++){
                        if(i==floor(p1/(n/G->K))){
                            p_1=p1%(n/G->K);
                        }
                        else{
                            p_1=0;
                        }
```

```
                    if(i==floor(p2/(n/G->K))){
                        p_2=p2%(n/G->K);
                    }
                    else{
                        p_2=(n/G->K)-1;
                    }
                    for(j=floor(q1/(n/G->K));j<=floor(q2/(n/G->K));j++){
                        if(j==floor(q1/(n/G->K))){
                            q_1=q1%(n/G->K);
                        }
                        else{
                            q_1=0;
                        }
                        if(j==floor(q2/(n/G->K))){
                            q_2=q2%(n/G->K);
                        }
                        else{
                            q_2=(n/G->K)-1;
                        }
                        range(G,n/G->K,p_1,p_2,q_1,q_2,dp+(n/G->K)*i,dq+(n/G->K)*j,y+G->K*i+j);
                    }
                }
            }
        }

        else{
            for(i=floor(p1/(n/G->K));i<=floor(p2/(n/G->K));i++){
                if(i==floor(p1/(n/G->K))){
                    p_1=p1%(n/G->K);
                }
                else{
                    p_1=0;
                }
                if(i==floor(p2/(n/G->K))){
                    p_2=p2%(n/G->K);
                }
                else{
                    p_2=(n/G->K)-1;
                }
                for(j=floor(q1/(n/G->K));j<=floor(q2/(n/G->K));j++){
                    if(j==floor(q1/(n/G->K))){
                        q_1=q1%(n/G->K);
                    }
                    else{
                        q_1=0;
                    }
                    if(j==floor(q2/(n/G->K))){
                        q_2=q2%(n/G->K);
                    }
                    else{
                        q_2=(n/G->K)-1;
                    }
                    range(G,n/G->K,p_1,p_2,q_1,q_2,dp+(n/G->K)*i,dq+(n/G->K)*j,y+G->K*i+j);
                }
            }
        }
    }
}
```

Για να πετύχουμε η συνάρτησή μας να δουλεύει όπως πρέπει και μετά τη συμπίεση των μονάδων προσθέσαμε το παρακάτω κομμάτι κώδικα :

```
flag=1;
/*dhladh auto o kwdikas me to pou vrethoun K*K mhdenika
(auto sumvainei giati kathe fora pou sunantame
ena -1 ston T_init emeis antikathisoume me monada to -1 kai vazoume gia
 paidia tou K*K mhdenika)
termatizei ki epistrefontai oi grammes ki oi sthles pou periexoun
assous,pou tha einai sunexomenes
afou o upopinakas mas einai gematos me assous*/
if(y<G->T_size){
      for (i=y; i<y+G->K*G->K; i++){
         if (H(G->T,i) == 1)
             flag = 0;
      }
      if (flag){/*tote olos o upopinakas  me riza T[z] tha periexei assous.
             Se auth thn periptwsh enas upopinakas
             periexei mono assous opote tha apo th thesi pou ksekinaei o
             upopinakas tha exoume se sunexomenes theseis
             predecessors kai successors*/
             range_all_one(G,n,p1,p2,q1,q2,dp,dq,z);
      }
      else{/*se periptwsh pou o upopinakas mas den einai gematos me assous sunexizetai h anadromi*/
         for(i=floor(p1/(n/G->K));i<=floor(p2/(n/G->K));i++){
             if(i==floor(p1/(n/G->K))){
                 p_1=p1%(n/G->K);
             }
             else{
                 p_1=0;
             }
             if(i==floor(p2/(n/G->K))){
                 p_2=p2%(n/G->K);
             }
             else{
                 p_2=(n/G->K)-1;
             }
             for(j=floor(q1/(n/G->K));j<=floor(q2/(n/G->K));j++){
                 if(j==floor(q1/(n/G->K))){
                     q_1=q1%(n/G->K);
                 }
                 else{
                     q_1=0;
                 }
                 if(j==floor(q2/(n/G->K))){
                     q_2=q2%(n/G->K);
                 }
                 else{
                     q_2=(n/G->K)-1;
                 }
                 range(G,n/G->K,p_1,p_2,q_1,q_2,dp+(n/G->K)*i,dq+(n/G->K)*j,y+G->K*i+j);
             }
         }
    }
}

else{
    for(i=floor(p1/(n/G->K));i<=floor(p2/(n/G->K));i++){
        if(i==floor(p1/(n/G->K))){
            p_1=p1%(n/G->K);
        }
        else{
            p_1=0;
        }
        if(i==floor(p2/(n/G->K))){
            p_2=p2%(n/G->K);
        }
        else{
            p_2=(n/G->K)-1;
        }
```

```
                for(j=floor(q1/(n/G->K));j<=floor(q2/(n/G->K));j++){
                    if(j==floor(q1/(n/G->K))){
                        q_1=q1%(n/G->K);
                    }
                    else{
                        q_1=0;
                    }
                    if(j==floor(q2/(n/G->K))){
                        q_2=q2%(n/G->K);
                    }
                    else{
                        q_2=(n/G->K)-1;
                    }
                    range(G,n/G->K,p_1,p_2,q_1,q_2,dp+(n/G->K)*i,dq+(n/G->K)*j,y+G->K*i+j);
                }
            }
        }
    }
}
```

Αν η μεταβλητή flag (που είναι μια βοηθητική μεταβλητή) είναι ίση με μονάδα, τότε ένας υποπίνακας είναι γεμάτος με μονάδες, οπότε στη λίστα μας πρέπει να τοποθετήσουμε όλες τις θέσεις των στηλών αλλά και των γραμμών από τις οποίες αποτελείται ο υποπίνακας που βρισκόμαστε. Πιο συγκεκριμένα πρέπει να «σπάσουμε» τον υποπίνακα στον οποίο βρισκόμαστε σε άλλους υποπίνακες ώστε να εξάγουμε όλες τις γραμμές και τις στήλες που έχουν μονάδες. Για αυτό τον λόγο δημιουργήσαμε μια επιπλέον συνάρτηση την range_all_one, της οποίας την υλοποίηση βλέπουμε παρακάτω.

```
/*Einai h sunarthsh pou dhmiourghsame gia na trexei anadromika se periptwsh
pou sthn range eimaste se enan upopinaka gemato me
monades kai theloume apo kathe grammh kai sthlh na eksagoume tis sthles
kai tis grammes antistoixa, pou periexoun assous*/
void range_all_one(compressed_graph_T * G,int n,int p1,int p2,int q1,int q2,int dp,int dq,int z){
    node_T * new_node;
    int i,j,y,flag,k,l;
    int p_1,p_2,q_1,q_2;

    if(z>=G->T_size){
            new_node=(node_T*)malloc(sizeof(node_T));
            if(new_node==NULL){
                printf("Error in range_all_one");
                exit(1);
            }
            new_node->column=dq;
            new_node->row=dp;
            new_node->next=G->head_dp_dq;
            G->head_dp_dq=new_node;
    }
    else{
           y=rank(G,z)*(G->K*G->K);
           for(i=floor(p1/(n/G->K));i<=floor(p2/(n/G->K));i++){
             if(i==floor(p1/(n/G->K))){
                 p_1=p1%(n/G->K);
             }
             else{
                 p_1=0;
             }
             if(i==floor(p2/(n/G->K))){
                 p_2=p2%(n/G->K);
             }
             else{
                 p_2=(n/G->K)-1;
             }
             for(j=floor(q1/(n/G->K));j<=floor(q2/(n/G->K));j++){
                 if(j==floor(q1/(n/G->K))){
                     q_1=q1%(n/G->K);
                 }
                 else{
                     q_1=0;
                 }
                 if(j==floor(q2/(n/G->K))){
                     q_2=q2%(n/G->K);
                 }
                 else{

                      q_2=(n/G->K)-1;
                  }
                  range_all_one(G,n/G->K,p_1,p_2,q_1,q_2,dp+(n/G->K)*i,dq+(n/G->K)*j,y+G->K*i+j);
              }
          }
    }
}
```

3.5.4 Link in range

Αυτή η συνάρτηση είναι μια γενικευμένη μορφή της συνάρτησης range, δηλαδή αντί να εξάγουμε τη θέση των στηλών ή των γραμμών που περιέχουν μονάδες ελέγχουμε αν ένας υποπίνακας περιέχει έστω και μια μονάδα (δηλαδή αν υπάρχει ακμή μεταξύ κάποιων κόμβων σε συγκεκριμένο εύρος τιμών), αν περιέχει, τότε η συνάρτηση link_in_range επιστρέφει την τιμή 1 αλλιώς την τιμή 0. Η συνάρτηση αυτή έχει αρχικές τιμές range(n',p_1, p_2,q_1, q_2,-1). Παρακάτω βλέπουμε και την υλοποίησή της.

```
/*me auth th sunarthsh elegxoume an uparxoun links metaksu komvwn metaksu twn
diasthmatwn [p1,p2] kai [q1,q2](p1,p1 oi grammes
kai q1,q1 oi sthles).An uparxei link tote h sunarthsh mas epistrefei 1 alliws 0 */
int link_in_range(compressed_graph_T G,int n,int p1,int p2,int q1,int q2,int z){
    int i,j,y;
    int p_1,p_2,q_1,q_2,flag;

    if(z>=G.T_size){
        return(H(G.L,(z-G.T_size)));
    }

    else{
        if(z==-1 || H(G.T,z) ) { /*to z pairnei thn timi -1 otan eimaste sthn riza
                 afou emeis ksekiname thn
                arithmisi twn level apo to 1 sto dendro mas*/
            if(p1==0 && q1==0  && p2==n-1 && q2==n-1){
                return 1;
            }
            y=rank(&G,z)*(G.K*G.K);
            flag = 1;
            /*dhladh me auton ton kwdika me to pou vrethoun K*K mhdenika
            termatizei ki epistrefei 1 giati o upopinakas tha einai olos me
            monades ara

            sigoura tha uparxei link metaksu twn komvwn sta diasthmata [p1,p2]
            kai [q1,q2]*/
            if(y<G.T_size){
                for (i=y; i<y+G.K*G.K; i++){
                    if (H(G.T,i) == 1)
                        flag = 0;
                }
                if (flag)
                    return(1);
            }
            for(i=floor(p1/(n/G.K));i<=floor(p2/(n/G.K));i++){
                if(i==floor(p1/(n/G.K))){
                    p_1=p1%(n/G.K);
                }
                else{
                    p_1=0;
                }
                if(i==floor(p2/(n/G.K))){
                    p_2=p2%(n/G.K);
                }
                else{
                    p_2=(n/G.K)-1;
                }
```

```
                for(j=floor(q1/(n/G.K));j<=floor(q2/(n/G.K));j++){
                    if(j==floor(q1/(n/G.K))){
                        q_1=q1%(n/G.K);
                    }
                    else{
                        q_1=0;
                    }
                    if(j==floor(q2/(n/G.K))){
                        q_2=q2%(n/G.K);
                    }
                    else{
                        q_2=(n/G.K)-1;
                    }
                    if(link_in_range(G,n/G.K,p_1,p_2,q_1,q_2,y+G.K*i+j)){
                        return 1;
                    }
                }
            }
        }
        return 0;
    }
}
```

Όπως και σε όλες τις προηγούμενες συναρτήσεις έτσι και σε αυτή αρχικά την είχαμε χτίσει έτσι ώστε να λειτουργεί για τη συμπίεση των μηδενικών μόνο. Για να δουλέψει και για τη συμπίεση των άσσων προσθέσαμε το παρακάτω κομμάτι κώδικα.

```
flag = 1;
/*dhladh me auton ton kwdika me to pou vrethoun K*K mhdenika
termatizei ki epistrefei 1 giati o upopinakas tha einai olos me
monades ara
sigoura tha uparxei link metaksu twn komvwn sta diasthmata [p1,p2]
kai [q1,q2]*/
if(y<G.T_size){
    for (i=y; i<y+G.K*G.K; i++){
        if (H(G.T,i) == 1)
            flag = 0;
    }
    if (flag)
        return(1);
}
```

Και πάλι η μεταβλητή flag παίρνει την τιμή 1 αν ένας ολόκληρος υποπίνακας είναι γεμάτος με άσσους αλλιώς παίρνει την τιμή 0. Όταν ένας ολόκληρος υποπίνακας είναι γεμάτος με μονάδες αρκεί να τερματίσουμε τη συνάρτησή μας και να μας επιστραφεί η τιμή 1, σε αντίθετη περίπτωση συνεχίζουμε την αναδρομή μέχρι να βρεθεί ένας άσσος στον υποπίνακά μας (δηλαδή να τερματίσει η συνάρτησή μας και να επιστραφεί η τιμή 1) ή σε περίπτωση που ολόκληρος ο υποπίνακάς μας είναι γεμάτος με μηδενικά να μας επιστραφεί η τιμή 0.

3.6 Main

Στη συνάρτηση main χτίζουμε τον πίνακα γειτνιάσεως, τον οποίο χρησιμοποιούμε για να εξάγουμε το k^2- δένδρο μας αλλά και για να ελέγξουμε την αποτελεσματικότητα των μεθόδων συμπίεσης που αναφέραμε. Σύμφωνα με την τιμή του k σπάμε τον πίνακά μας σε υποπίνακες, την τιμή της μεταβλητής αυτής την εισάγει ο χρήστης. Επίσης, εισάγει την τιμή της μεταβλητής X (ανά πόσες θέσεις στον πίνακα acc αποθηκεύουμε το πλήθος των μονάδων του T) αλλά και το μέγεθος του πίνακα γειτνιάσεως (n). Για τον πίνακα γειτνιάσεως κάνουμε δυναμική εκχώρηση μνήμης και στη συνέχεια τον «γεμίζουμε» με τυχαία στοιχεία (0 ή 1) με τη χρήση της συνάρτησης rand [35]. Σε περίπτωση που το μέγεθος του πίνακά μας δεν είναι κάποια δύναμη του k, τότε χρειάζεται να κάνουμε επέκταση του πίνακα. Δηλαδή θα γεμίσουμε τον πίνακά μας με μηδενικά στα δεξιά και στο κάτω μέρος του. Θα του προσθέσουμε τόσες γραμμές και στήλες ώστε το μέγεθος του να γίνει κάποια δύναμη του k. Επιπλέον, στη συνάρτηση main προσθέσαμε κώδικα ώστε η δημιουργία του πίνακα γειτνιάσεως να γίνεται μόνο μια φορά και για τις διάφορες τιμές του k ο ίδιος πίνακας να επεκτείνεται όπως έχουμε περιγράψει σε προηγούμενα κεφάλαια εκτενέστατα.

Όπως έχουμε αναφέρει παραπάνω δε μας ενδιαφέρει ο χρόνος που παίρνει η διαδικασία χτισίματος του πίνακα γειτνιάσεως αλλά μας ενδιαφέρει ο χρόνος που γίνονται η συμπίεση των μονάδων και των άσσων, ο χρόνος χτισίματος των πινάκων T και L αλλά κι ο χρόνος εκτέλεσης των ερωτημάτων που θέσαμε. Για να μετρήσουμε τον χρόνο χρησιμοποιήσαμε τη συνάρτηση gettimeofday [33,37] από τη βιβλιοθήκη της c, η οποία είναι μια συνάρτηση που μας προσφέρει μεγάλη ακρίβεια. Για τη χρήση της ήταν απαραίτητη η δημιουργία ενός struct, το οποίο βλέπουμε παρακάτω.

```
/*auto einai to struct pou periexei tis metavlites gia kathe sunarthshs
 pou theloume na metrhsoume ton xrono*/
struct timeval check_link1,check_link2,successors1,successors2,predecessors1;
struct timeval range1_64,range2_64,link_in_range1_64;
struct timeval predecessors2,range1,range2,link_in_range1;
struct timeval link_in_range2,create_graph1,create_graph2,link_in_range2_64;
```

Είναι σημαντικό να αναφέρουμε πως στη συνάρτηση main χρησιμοποιήσαμε μια επανάληψη, ώστε για τον ίδιο πίνακα γειτνιάσεως να εκτελούνται οι συναρτήσεις μας για διαφορετικό k (θα αναφερθούμε εκτενέστερα στα πειράματα που διενεργήσαμε στο επόμενο κεφάλαιο). Παρακάτω βλέπουμε και την υλοποίηση της συνάρτησης main.

```c
int main(int argc,char * argv[]){
    compressed_graph_T  G;
    int **A;
    int N;
    time_t t;
    int k,i,j,l;
     /*auto einai to struct pou periexei tis metavlites gia kathe sunarthshs
      pou theloume na metrhsoume ton xrono*/
    struct timeval check_link1,check_link2,successors1,successors2,predecessors1;
    struct timeval range1_64,range2_64,link_in_range1_64;
    struct timeval predecessors2,range1,range2,link_in_range1;
    struct timeval link_in_range2,create_graph1,create_graph2,link_in_range2_64;

    printf("Give a number for N:\n");/*opou N einai to megethos tou pinaka
    prin thn epektash,to vazoume emeis*/
    scanf("%d",&N);
    printf("Give a number for K:\n");
    scanf("%d",&G.K);
    printf("Give a number for X:\n");/*opou X einai ana poses theseis
    tha apothikeuoume ston pinaka acc to plithos
    twn monadwn (rank)*/
    scanf("%d",&G.X);

     G.pinakas_geitniasews_initial=(int**)malloc(N*sizeof(int*));
     if(G.pinakas_geitniasews_initial==NULL){
            printf("Error in pinakas_geitniasews_initial");
            exit(1);
    }
    for(i=0;i<N;i++){
        G.pinakas_geitniasews_initial[i]=(int*)malloc(N*sizeof(int));
        if(G.pinakas_geitniasews_initial[i]==NULL){
            printf("Error in pinakas_geitniasews_initial");
            exit(1);
        }
    }


    srand(time(NULL));
    //xtisimo enos pinaka NxN (afou megethos=N) me tuxaia stoixeiá apo
     //to 0 nexri to 2 xwris to 2
    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
            G.pinakas_geitniasews_initial[i][j]=rand()%2;
        }
    }
   for(l=0;l<3;l++){/*edw vazoume autes tis times gia na kanoume dokimes
        gia ton idio pinaka geitniasews alla diaforetiko K=2,3,4*/

        printf ("\t\tEXPERIMENT: K=%d , N=%d\n\n",G.K,N);

        G.N_new=extension(G.K,N);

        //desmeusi xwrou gia pinaka N_new x N_new ki antigrafh tou N x N pinaka
         ston N_new x N_new ki analogh epektash
        G.pinakas_geitniasews=(int**)malloc(G.N_new*sizeof(int*));
        if(G.pinakas_geitniasews==NULL){
                printf("Error in pinakas_geitniasews");
                exit(1);
        }
        for(i=0;i<G.N_new;i++){
            G.pinakas_geitniasews[i]=(int*)malloc(G.N_new*sizeof(int));
            if(G.pinakas_geitniasews[i]==NULL){
                printf("Error in pinakas_geitniasews");
                exit(1);
            }
        }
```

```
for(i=0;i<N;i++){
    for(j=0;j<N;j++){
        G.pinakas_geitniasews[i][j]=G.pinakas_geitniasews_initial[i][j];
    }
}

 /*an h epektasi einai 1 tote to N(megethos) einai dunami tou k opote
  to G->N_new =megethos, diladi o pinakas mas de xreiazetai epektash*/
if (G.N_new!=N){
   for(i=0;i<G.N_new;i++){
        for(j=N;j<G.N_new;j++){
            G.pinakas_geitniasews[i][j]=0;
        }
    }

    for(i=N;i<G.N_new;i++){
        for(j=0;j<G.N_new;j++){
            G.pinakas_geitniasews[i][j]=0;
        }
    }
}

gettimeofday(&create_graph1, NULL);/*upologismos xronou treksimatos ths
 create graph*/
create_graph(&G,N);

gettimeofday(&create_graph2, NULL);

printf ("Time of create_graph=%f seconds\n\n",
        (double) (create_graph2.tv_usec - create_graph1.tv_usec) / 1000000 +
        (double) (create_graph2.tv_sec - create_graph1.tv_sec));

srand((unsigned) time(&t)); /*Intializes random number generator*/

/*edw kaloume thn ckeck_link me dikes mas parametrous,h teleutaia parametros
 kseikinaei apo -1 afou emeis den exoume level 0 sto dendro mas alla
  ksekiname apo l=1*/

gettimeofday(&check_link1, NULL); /*upologismos xronou treksimatos ths ckeck link*/

/*Gia th sunarthsh check_link tha paroume (10/100)*N theseis (grammes kai sthles)
gia na vgaloume sumperasma gia ton xrono*/
for(k=0;k<(int)(0.1*N);k++){
    i=rand()%N;
    j=rand()%N;
    check_link(G,G.N_new,i,j,-1);
}
gettimeofday(&check_link2, NULL);

printf("Ammount of cells  :%d , Time of check_link  =%f seconds\n\n",(int)(0.1*N),
        (double) (check_link2.tv_usec - check_link1.tv_usec) / 1000000 +
        (double) (check_link2.tv_sec - check_link1.tv_sec));

/*Mesa stis sunarthseis tha ftiaksoume kwdika wste na ginontai epanalhpseis
 gia diafora diasthmata wste na pairnoun tuxaies times
kai tha doulepsoume me thn srand */

gettimeofday(&successors1, NULL); /*upologismos xronou treksimatos ths
 successors*/

for(k=0;k<(int)(0.1*N);k++){/*Gia th sunarthsh successors tha paroume (10/100)*N
        theseis (grammes) gia na vgaloume
        sumperasma gia ton xrono*/
    G.s_head=NULL;/*arxikopoihsh ths listas pou periexei tous succcessors*/
    i=rand()%N;
    successors(&G,G.N_new,i,0,-1);
    G.s_head=clear_list(G.s_head);
}

gettimeofday(&successors2, NULL);
```

```
printf ("Ammount of rows   :%d , Time of successors  =%f seconds\n\n",(int)(0.1*N),
            (double) (successors2.tv_usec - successors1.tv_usec) / 1000000 +
            (double) (successors2.tv_sec - successors1.tv_sec));

gettimeofday(&predecessors1, NULL);/*upologismos xronou treksimatos
ths predecessors*/

for(k=0;k<(int)(0.1*N);k++){/*Gia th sunarthsh predecessors tha paroume
        (10/100)*N theseis (grammes)
    gia na vgaloume sumperasma gia ton xrono*/
    G.p_head = NULL;/*arxikopoihsh ths listas pou periexei
tous predecessors*/
    i=rand()%N;
    predecessors(&G,G.N_new,i,0,-1);
    G.p_head = clear_list(G.p_head);
}

gettimeofday(&predecessors2, NULL);

printf ("Ammount of columns:%d , Time of predecessors=%f seconds\n\n",(int)(0.1*N),
            (double) (predecessors2.tv_usec - predecessors1.tv_usec) / 1000000 +
            (double) (predecessors2.tv_sec - predecessors1.tv_sec));

/*Gia th sunasthrh range tha kanoume duo diaforetika peiramata to ena
tha afora tetragwnikous pinakes opou oi diastaseis
tous tha einai 4% tou N kai tha topothetithoun se 8% tou N tuxaies theseis
 mesa ston NxN pinaka mas.*/

gettimeofday(&range1, NULL);/*upologismos xronou treksimatos ths range*/

for(k=0;k<(int)(0.08*N);k++){
    G.head_dp_dq = NULL;/*arxikopoihsh ths listas pou periexei tous komvous
    pou periexoun assous se sugkekrimena diasthmata*/
    i=rand()%(N-(int)(0.04*N));
    j=rand()%(N-(int)(0.04*N));
    range(&G,G.N_new,i,i+(int)(0.04*N),j,j+(int)(0.04*N),0,0,-1);/*to euros tou
    diasthmatos tha einai to 4% tou N*/
    G.head_dp_dq = clear_list(G.head_dp_dq);
}

gettimeofday(&range2, NULL);

printf("Range 4%% of N and 8%% of N:\n");

printf ("Size of submatrix: %d x %d , Ammount of submatrices: %d , Time of range=%f seconds\n\n",
        (int)(0.04*N),(int)(0.04*N),
        (int)(0.08*N),(double) (range2.tv_usec - range1.tv_usec) / 1000000 +
        (double) (range2.tv_sec - range1.tv_sec));

/*To deutero peirama tha afora tetragwnikous pinakes opou oi
diastaseis tous tha einai 64% tou N kai tha topothetithoun se
1% tou N tuxaies theseis mesa ston NxN pinaka mas*/

gettimeofday(&range1_64, NULL);/*upologismos xronou treksimatos
 ths range me upopinakes 64%x64%*/

for(k=0;k<(int)(0.01*N);k++){
    G.head_dp_dq = NULL;/*arxikopoihsh ths listas pou periexei tous
     komvous pou periexoun assous se sugkekrimena diasthmata*/
    i=rand()%(N-(int)(0.64*N));
    j=rand()%(N-(int)(0.64*N));
    range(&G,G.N_new,i,i+(int)(0.64*N),j,j+(int)(0.64*N),0,0,-1);/*to euros tou
    diasthmatos tha einai to 64% tou N*/
    G.head_dp_dq = clear_list(G.head_dp_dq);
}

gettimeofday(&range2_64, NULL);

printf("Range 64%% of N and 1%% of N:\n");
```

```
        printf("Size of submatrix: %d x %d , Ammount of submatrices: %d , Time of range=%f seconds\n\n",
               (int)(0.64*N),(int)(0.64*N),
                (int)(0.01*N),(double) (range2_64.tv_usec - range1_64.tv_usec) / 1000000 +
                (double) (range2_64.tv_sec - range1_64.tv_sec));

        /*Gia th sunasthrh link_in_range tha kanoume duo diaforetika peiramata to
        ena tha afora tetragwnikous pinakes opou oi diastaseis
        tous tha einai 4% tou N kai tha topothetithoun se 8% tou N tuxaies theseis
         mesa ston NxN pinaka mas.*/

        gettimeofday(&link_in_range1, NULL);/*upologismos xronou treksimatos ths
        range*/

        for(k=0;k<(int)(0.08*N);k++){
            i=rand()%(N-(int)(0.04*N));
            j=rand()%(N-(int)(0.04*N));
            link_in_range(G,G.N_new,i,i+(int)(0.04*N),j,j+(int)(0.04*N),-1);
            /*to euros tou diasthmatos tha einai to 4% tou N*/
        }

        gettimeofday(&link_in_range2, NULL);

        printf("Link in range 4%% of N and 8%% of N:\n");

        printf ("Size of submatrices: %d x %d , Ammount of submatrices: %d , Time of link_in_range=%f seconds\n\n",
                (int)(0.04*N),(int)(0.04*N),
                (int)(0.08*N),(double) (link_in_range2.tv_usec - link_in_range1.tv_usec) / 1000000 +
                (double) (link_in_range2.tv_sec - link_in_range1.tv_sec));

        /*To deutero peirama tha afora tetragwnikous pinakes opou oi diastaseis tous
        tha einai 64% tou N kai tha topothetithoun se
        1% tou N tuxaies theseis mesa ston NxN pinaka mas*/

        gettimeofday(&link_in_range1_64, NULL);/*upologismos xronou treksimatos ths range me upopinakes 64%x64%*/

        for(k=0;k<(int)(0.01*N);k++){
            i=rand()%(N-(int)(0.64*N));
            j=rand()%(N-(int)(0.64*N));
            link_in_range(G,G.N_new,i,i+(int)(0.64*N),j,j+(int)(0.64*N),-1);/*to euros tou diasthmatos
             tha einai to 64% tou N*/
        }

        gettimeofday(&link_in_range2_64, NULL);

        printf("Link in range 64%% of N and 1%% of N:\n");


        printf("Size of submatrices: %d x %d , Ammount of submatrices: %d , Time of link_in_range=%f seconds\n\n",
               (int)(0.64*N),(int)(0.64*N),
                (int)(0.01*N),(double) (link_in_range2_64.tv_usec - link_in_range1_64.tv_usec) / 1000000 +
                (double) (link_in_range2_64.tv_sec - link_in_range1_64.tv_sec));
         G.K++;
         free(G.T);
         free(G.L);

    }

     return 0;
}
```

4. ΠΕΙΡΑΜΑΤΑ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

4.1 Γενικά

Σε αυτό το κεφάλαιο θα παρουσιάσουμε με πίνακες τα αποτελέσματα των πειραμάτων που διεξαγάγαμε. Επικεντρωνόμαστε στο χρόνο που χρειάζεται κάθε συνάρτηση για να εκτελεστεί σύμφωνα με τις διάφορες τιμές για τις παραμέτρους που εισάγουμε. Η συνάρτηση gettimeofday μετράει τον χρόνο εκτέλεσης σε δευτερόλεπτα, για μεγαλύτερη ακρίβεια. Είναι αναγκαίο να επισημάνουμε πως πραγματοποιήσαμε πειράματα για τις εξής τιμές του n =900,2048 (μέγεθος του πίνακα γειτνιάσεως) και για τα αντίστοιχα k=2,3,4.

Κάθε συνάρτηση στη main (στην προηγούμενη παράγραφο βλέπουμε και την υλοποίησή της) την καλέσαμε για διάφορες τιμές στις παραμέτρους της, για να μπορέσουμε να ελέγξουμε όσο γίνεται καλύτερα τη χρονική απόδοση των συμπιέσεων που πραγματοποιήσαμε. Κάθε μια συνάρτηση την τοποθετήσαμε σε μια επανάληψη (for), ώστε για τις ίδιες τιμές του k να καλείται με διαφορετικές παραμέτρους. Πιο συγκεκριμένα τη συνάρτηση create_graph την καλέσαμε για n=900,2048 και k=2,3,4. Για τη συνάρτηση ckeck_link στις παραμέτρους p και q τοποθετήσαμε τυχαίες τιμές με τη συνάρτηση rand και επαναλάβαμε την εκτέλεσή της τόσες φορές όσο το δέκα τοις εκατό του n. Σε κάθε επανάληψή της οι τιμές των p και q αλλάζουν. Για την καλύτερη κατανόησή της παραπάνω διαδικασίας θα αναφερθούμε σε ένα παράδειγμα. Αν για παράδειγμα το n είναι ίσο με 100, τότε θα καλέσουμε τη συνάρτηση ckeck_link 10 φορές με τυχαίες τιμές των p και q. Σε κάθε συνάρτηση μετράμε τον χρόνο αφού γίνουν όλες οι εκτελέσεις που επιθυμούμε. Στην προηγούμενη παράγραφο βλέπουμε το κομμάτι του κώδικα που αφορά τις εκτελέσεις της check_link (παράγραφος 3.6). Όπως την check_link έτσι και τις συναρτήσεις successors και predecessors τις καλέσαμε τόσες φορές όσο και το δέκα τοις εκατό του n. Κι οι παράμετροι p και q στις συναρτήσεις successors και predecessors αντίστοιχα παίρνουν τυχαίες τιμές για όλες τις επαναλήψεις ((10/100)*n), όπως μπορούμε να δούμε και παραπάνω (παράγραφος 3.6) στην υλοποίησή τους. Επίσης, στην αρχή κάθε επανάληψης

αρχικοποιούμε τις κεφαλές των λιστών που θα αποθηκευτούν τα αποτελέσματα των συναρτήσεων successors και predecessors και στο τέλος κάθε επανάληψης αποδεσμεύουμε όλη την μνήμη που δεσμεύσαμε για τις λίστες μας ώστε στην επόμενη επανάληψη να έχουμε αρκετό χώρο για να χτιστούν ξανά οι λίστες μας.

Για τις συναρτήσεις range και link_in_range θα πραγματοποιήσουμε ερωτήματα για μεγέθη υποπινάκων 4% x 4% και 64% x 64% του n τοποθετημένων σε τυχαίες θέσεις του πίνακα, όπου το πλήθος των θέσεων να είναι 8% και 1% του n αντίστοιχα, παραπάνω βλέπουμε και τη υλοποίησή τους (παράγραφος 3.6). Αν για παράδειγμα πραγματοποιήσουμε το πείραμα για το 4 τοις εκατό του n και το n ισούται με 1000, τότε τα πειράματα για τις range και link_in_range θα αφορούν πίνακες 40 x 40 που θα τοποθετηθούν σε 80 τυχαίες θέσεις μέσα στον πίνακα μεγέθους 1000 x 1000. Όπως και στις συναρτήσεις successors και predecessors έτσι και στην range στην αρχή κάθε επανάληψης αρχικοποιούμε την κεφαλή της λίστας που θα αποθηκευτούν τα αποτελέσματα και στο τέλος κάθε επανάληψης αποδεσμεύουμε όλη την μνήμη που δεσμεύσαμε για τη λίστα αυτή.

4.2 Παρουσίαση αποτελεσμάτων

Στην προηγούμενη παράγραφο περιγράψαμε τα πειράματα που πραγματοποιήσαμε. Σε αυτή την παράγραφο θα παρουσιάσουμε τα αποτελέσματα σε πίνακες αλλά θα τα παραστήσουμε και γραφικά, δηλαδή πόσο χρόνο διήρκησαν οι εκτελέσεις κάθε συνάρτησης. Τα πειράματα που πραγματοποιήσαμε είχαν τις εξής τιμές: για n=900,2048 και k=2,3,4. Είναι σημαντικό να αναφέρουμε πως τα πειράματα έγιναν σε έναν τυπικό υπολογιστή με συνηθισμένες δυνατότητες, με αποτέλεσμα να μην μπορούμε να κάνουμε πειράματα με πολύ μεγάλα n, λόγω έλλειψης χώρου στη μνήμη ram (4 gb). Επιπλέον, όλοι οι χρόνοι που αναφέρονται παρακάτω είναι μετρημένοι σε δευτερόλεπτα (seconds).

k \ **n**	**900**	**2048**
2	14.095109	445.593471
3	0.502005	8.950072
4	0.19001	1.112004

Πίνακας 1: Χρονικά αποτελέσματα της συνάρτησης create_graph

Για n = 900 για τη συνάρτηση check_link ελέγχθηκαν 90 τυχαία κελιά (αν υπάρχει 1 ή 0,δηλαδή αν υπάρχει ακμή ή όχι αντίστοιχα), ενώ για n = 2048 ελέγχθηκαν 204 τυχαία κελιά του πίνακα γειτνιάσεως.

k \ **n**	**900**	**2048**
2	0	0.001
3	0	0
4	0	0

Πίνακας 2: Χρονικά αποτελέσματα της συνάρτησης check_link

Για n = 900 εξάχθηκαν οι successors 90 γραμμών και για n = 2048 εξάχθηκαν οι successors 204 γραμμών.

k \ **n**	**900**	**2048**
2	0.040000	0.218013
3	0.030000	0.144003
4	0.020000	0.130000

Πίνακας 3: Χρονικά αποτελέσματα της συνάρτησης successors

Για n = 900 εξάχθηκαν οι predecessors 90 στηλών και για n = 2048 εξάχθηκαν οι predecessors 204 στηλών.

k \ **n**	**900**	**2048**
2	0.040000	0.267015
3	0.030000	0.150000
4	0.030000	0.120000

Πίνακας 4: Χρονικά αποτελέσματα της συνάρτησης predecessors

Στον παρακάτω πίνακα βλέπουμε τα πειράματα, για n = 900 όπου ελέγχθηκαν πίνακες μεγέθους 36 x 36 (το 4% του n), οι οποίοι τοποθετήθηκαν σε 72 (το 8% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως. Για n = 2048 ελέγχθηκαν πίνακες μεγέθους 81 x 81 (το 4% του n), οι οποίοι τοποθετήθηκαν σε 163 (το 8% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως.

k \ **n**	**900**	**2048**
2	0.110000	1.228058
3	0.080000	0.880001
4	0.070000	0.783005

Πίνακας 5: Χρονικά αποτελέσματα της συνάρτησης range (4% του n και 8% του n)

Στον παρακάτω πίνακα βλέπουμε τα πειράματα, για n = 900 όπου ελέγχθηκαν πίνακες μεγέθους 576 x 576 (το 64% του n), οι οποίοι τοποθετήθηκαν σε 9 (το 1% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως. Για n = 2048 ελέγχθηκαν πίνακες μεγέθους 1310 x 1310 (το 64% του n), οι οποίοι τοποθετήθηκαν σε 20 (το 1% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως.

k \ **n**	**900**	**2048**
2	3.148012	36.764358
3	2.390022	27.541282
4	2.038008	23.826133

Πίνακας 6: Χρονικά αποτελέσματα της συνάρτησης range (64% του n και 1% του n)

Στον παρακάτω πίνακα βλέπουμε τα πειράματα, για n = 900 όπου ελέγχθηκαν πίνακες μεγέθους 36 x 36 (το 4% του n), οι οποίοι τοποθετήθηκαν σε 72 (το 8% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως. Για n = 2048 ελέγχθηκαν πίνακες μεγέθους 81 x 81 (το 4% του n), οι οποίοι τοποθετήθηκαν σε 163 (το 8% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως.

k \ n	900	2048
2	0	0
3	0	0
4	0	0

Πίνακας 7: Χρονικά αποτελέσματα της συνάρτησης link_in_range (4% του n και 8% του n)

Στον παρακάτω πίνακα βλέπουμε τα πειράματα, για n = 900 όπου ελέγχθηκαν πίνακες μεγέθους 576 x 576 (το 64% του n), οι οποίοι τοποθετήθηκαν σε 9 (το 1% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως. Για n = 2048 ελέγχθηκαν πίνακες μεγέθους 1310 x 1310 (το 64% του n), οι οποίοι τοποθετήθηκαν σε 20 (το 1% του n) τυχαίες θέσεις μέσα στον πίνακα γειτνιάσεως.

k \ n	900	2048
2	0	0
3	0	0
4	0	0

Πίνακας 8: Χρονικά αποτελέσματα της συνάρτησης link_in_range (64% του n και 1% του n)

4.3 Συμπεράσματα των πειραμάτων

Σε αυτή την υποπαράγραφο θα αναλύσουμε τα αποτελέσματα που έχουμε εξάγει από τα διάφορα ερωτήματα που έχουμε πραγματοποιήσει (τα αποτελέσματα τα βλέπουμε στην προηγούμενη παράγραφο). Θα αναλύσουμε ξεχωριστά τα αποτελέσματα κάθε συνάρτησης. Όπως έχουμε αναφέρει στις προηγούμενες παραγράφους εμείς κάναμε πειράματα για δύο διαφορετικά μεγέθη πινάκων (900,2048) και τρεις διαφορετικές τιμές του k (2,3,4). Θα συγκρίνουμε τους χρόνους που έχουμε για όλες τις διαφορετικές τιμές. Επίσης, είναι σημαντικό να αναφέρουμε πως για n = 900 και n = 2048 οι πίνακες που κάνουμε πειράματα είναι διαφορετικοί, απλά για n = 900 και k = 2, 3, 4 επεξεργαζόμαστε τον ίδιο πίνακα αλλά διαφορετικά επεκταμένο (έχουμε αναφερθεί σε προηγούμενες παραγράφους εκτενέστατα), το ίδιο ακριβώς ισχύει και για n = 2048.

Ξεκινώντας από τη συνάρτηση create_graph, που είναι η βασική μας συνάρτηση αφού εκεί γίνεται η δημιουργία των πινάκων T και L (που είναι οι πίνακες στους οποίους βασίζονται οι μέθοδοι συμπίεσης που χρησιμοποιούμε). Παρατηρούμε στον πίνακα 1 ότι για μεγαλύτερα μεγέθη πινάκων αυξάνεται ο χρόνος εκτέλεσης της create_graph που είναι λογικό αφού όσο πιο μεγάλος πίνακας, τόσο πιο πολλά στοιχεία για έλεγχο έχουμε σε κάθε υποπίνακα. Επίσης, βλέπουμε πως όσο μεγαλώνει το k, τόσο πιο λίγος χρόνος χρειάζεται για να εκτελεστεί η συνάρτηση create_graph. Τεράστια διαφορά μπορούμε να παρατηρήσουμε για n = 2048, όπου για k=2 ο χρόνος εκτέλεσης είναι 445.593471 sec ενώ για k = 3 ο χρόνος εκτέλεσης είναι 8.950072 sec. Αυτό οφείλεται στο γεγονός πως όσο μεγαλύτερο είναι το k, τόσο λιγότερα επίπεδα θα έχουμε στο k^2- δένδρο μας, άρα κατ'επέκταση περισσότερες αναζητήσεις σε κάθε επίπεδο του δένδρου αλλά λιγότερες στο σύνολο των επιπέδων.

Η check_link είναι η συνάρτηση με την οποία ελέγχουμε αν υπάρχει κάποια ακμή (σύνδεσμος) μεταξύ δύο κόμβων. Όπως βλέπουμε στον πίνακα 2 οι χρόνοι εκτέλεσης τη συγκεκριμένης συνάρτησης είναι σχεδόν μηδενικοί, εκτός αυτών για k = 2 και n = 2048. Από αυτή τη συνάρτηση λοιπόν δε μπορεί να βγει ένα ξεκάθαρο αποτέλεσμα για το πιο k είναι πιο αποδοτικό.

Με τη συνάρτηση successors εξάγουμε από κάποιες σειρές τις θέσεις των στηλών που περιέχουν μονάδες. Στον πίνακα 3 βλέπουμε πως όσο πιο μεγάλο k έχουμε τόσο πιο λίγος χρόνος χρειάζεται για να ολοκληρωθεί η εξαγωγή όλων των successors των σειρών που εμείς έχουμε ορίσει, αυτό το παρατηρούμε για n = 900 αλλά και για n = 2048. Βέβαια οι χρόνοι εκτέλεσης για τα διάφορα k δε διαφέρουν και πολύ μεταξύ τους. Παρόλα αυτά και για αυτή τη συνάρτηση μπορούμε να βγάλουμε συμπέρασμα πως για μεγαλύτερα k οι χρόνοι εκτέλεσης είναι καλύτεροι.

Με τη συνάρτηση predecessors εξάγουμε από κάποιες στήλες τις θέσεις των γραμμών που περιέχουν μονάδες. Όπως και στη συνάρτηση successors έτσι και σε αυτή (πίνακας 4) παρατηρούμε ότι όσο μεγαλύτερες τιμές δίνουμε στο k (για n = 900 και για n = 2048) τόσο λιγότερο χρόνο χρειάζεται για να εκτελεστεί η συνάρτηση predecessors. Οπότε όπως και στις προηγούμενες συναρτήσεις έτσι και σε αυτή μπορούμε να εξάγουμε το συμπέρασμα πως για μεγαλύτερα k (και το ίδιο) n ο χρόνος εκτέλεσής τους βελτιώνεται.

Με τη συνάρτηση range εξάγουμε τις στήλες και τις γραμμές, σε συγκεκριμένο εύρος, που περιέχουν μονάδες. Για τη συνάρτηση range πραγματοποιήσαμε πειράματα για δύο διαφορετικά μεγέθη υποπινάκων (64% του n και 4% του n) , που ήταν τοποθετημένοι σε τυχαίες θέσεις (1% του n και 8% του n) στον πίνακα γειτνιάσεως. Σε όλες τις περιπτώσεις αλλά και για τα δύο μεγέθη (n = 900 και n = 2048) που διεξαγάγαμε τα πειράματα (πίνακες 5,6) βλέπουμε πως όσο μεγαλώνουμε το k τόσο μικραίνουν οι χρόνοι εκτέλεσης της συγκεκριμένης συνάρτησης.

Η συνάρτηση link_in_range είναι η συνάρτηση, με την οποία ελέγχουμε αν σε έναν υποπίνακα (με συγκεκριμένο εύρος τιμών) υπάρχει τουλάχιστον μια μονάδα. Από ότι βλέπουμε στους πίνακες 7 και 8, για τις διάφορες τιμές του k ο χρόνος εκτέλεσης είναι μηδενικός. Πράγμα που είναι πολύ λογικό αφού η συνάρτηση αυτή είναι φτιαγμένη με τέτοιον τρόπο έτσι ώστε με το που βρεθεί ο πρώτος άσσος σε έναν υποπίνακα να τερματίζει και να επιστρέφεται η τιμή 1. Αν ένας υποπίνακας είναι γεμάτος με μηδενικά, τότε θα απεικονίζεται στο δένδρο με ένα μοναδικό μηδενικό και σε αυτήν την περίπτωση η συνάρτησή μας θα τερματίζει και θα επιστρέφεται η τιμή 0, μια εξίσου γρήγορη διαδικασία. Ενδεχομένως, αν ο τυπικός υπολογιστής που χρησιμοποιήσαμε είχε καλύτερες δυνατότητες, και μπορούσαμε να «τεστάρουμε» τις συναρτήσεις μας για μεγαλύτερα n, ίσως να φαινόταν και σε αυτή τη συνάρτηση η διαφορά για μεγαλύτερα k.

Τελικά, μπορούμε να βγάλουμε το συμπέρασμα πως όσο μεγαλύτερο k έχουμε τόσο καλύτερες χρονικές αποδόσεις μπορούμε να πετύχουμε.

5. ΑΞΙΟΛΟΓΗΣΗ, ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΕΠΕΚΤΑΣΕΙΣ

5.1 Συνολικά συμπεράσματα της εργασίας

Στην παρούσα διπλωματική εργασία παρουσιάστηκε μια νέα συμπαγής αναπαράσταση γραφημάτων, το k^2- δένδρο. Αυτή η αναπαράσταση εκμεταλλεύεται το γεγονός πως μεγάλες περιοχές του πίνακα γειτνιάσεως μπορούν να περιέχουν μηδενικά ή μονάδες. Για την «εκμετάλλευσή» των μεγάλων αυτών περιοχών δημιουργήσαμε δύο μεθόδους συμπίεσης, τη συμπίεση των μονάδων και των μηδενικών. Η συγκεκριμένη συμπαγής αναπαράσταση γραφημάτων υποστηρίζει ποικίλα ερωτήματα (queries), όπως ο έλεγχος ύπαρξης ακμής μεταξύ δύο κόμβων, η εύρεση των successors και predecessors ενός κόμβου κι άλλα πιο γενικευμένα ερωτήματα, όπως αν υπάρχει έστω και μια ακμή μεταξύ ενός εύρους κόμβων.

Με τις μεθόδους που αναπτύξαμε καταφέραμε να «χωρέσουμε» πολύ μεγάλα γραφήματα στην κύρια μνήμη ενός τυπικού υπολογιστή (όπως αυτός που χρησιμοποιήθηκε για την υλοποίηση της συγκεκριμένης εργασίας) αλλά και να απαντηθούν κάποια ερωτήματα σε σχετικά καλό χρόνο. Όπως έχουμε αναφέρει η συγκεκριμένη συμπαγής αναπαράσταση είναι εξίσου αποδοτική στην προς τα εμπρός αλλά και προς τα πίσω αναζήτηση (navigation).

5.2 Βήματα για την ολοκλήρωση της εργασίας

Αρχικά χτίσαμε τον πίνακα γειτνιάσεως στη συνάρτηση main και στη συνέχεια δημιουργήσαμε τις συναρτήσεις που βοηθούν στην επέκταση (extension και power) του πίνακά μας (πίνακας γειτνιάσεως). Στη συνέχεια προχωρήσαμε στη δημιουργία της build, όπου αρχικά εκτελούσαμε μόνο συμπίεση των μηδενικών σε αυτή. Μετέπειτα προχωρήσαμε στη δημιουργία των διάφορων ερωτημάτων, για να αξιολογήσουμε τη μέθοδο συμπίεσης που δημιουργήσαμε. Αφού

ολοκληρώσαμε όλες τις συναρτήσεις αλλά και τα ερωτήματα θελήσαμε να εμπλουτίσουμε τη συμπαγή αναπαράστασή μας και με τη συμπίεση των μονάδων. Μετά τη συμπίεση των μονάδων χρειάστηκαν αρκετές επιπλέον μετατροπές στην build αλλά και σε όλα τα υπόλοιπα ερωτήματα που είχαμε θέσει, όπως αναφέρουμε και σε προηγούμενες παραγράφους. Τέλος, λόγω του μεγάλου όγκου του κώδικα που είχε γραφτεί ήταν αναγκαίος ο διαχωρισμός του σε τρία κομμάτια, όπως αναφέρουμε και στην παράγραφο 3.1.

5.3 Προτάσεις για μελλοντικές επεκτάσεις

Όπως έχουμε αναφέρει και σε προηγούμενες παραγράφους η συγκεκριμένη συμπαγής αναπαράσταση γραφημάτων μπορεί να χρησιμοποιηθεί για διάφορα σενάρια. Θα μπορούμε για παράδειγμα ο πίνακας γειτνιάσεως να αφορά περιοχές που είναι μολυσμένες ή μη (εκτενέστερη αναφορά στην παράγραφο 2.1). Ένα άλλο παράδειγμα που θα μπορούσε να αναπαριστά ο πίνακας γειτνιάσεως είναι περιοχές που περιέχουν δένδρα. Αν έχουμε μια περιοχή μόνο με μηδενικά, τότε δε θα υπάρχει κανένα δένδρο, αν έχουμε μόνο άσσους τότε τα δένδρα είναι πολλά σε αυτήν την περιοχή και τέλος αν έχουμε μηδενικά κι άσσους, τότε έχουμε λίγα δένδρα. Η παραπάνω εφαρμογή του πίνακα γειτνιάσεως θα μπορούσε να υλοποιηθεί από τους δήμους κάθε πόλης, ώστε να γίνεται ανάλογη φύτευση δένδρων σε μια πόλη. Βέβαια υπάρχουν πολλά ακόμη πεδία που θα μπορούσε να εφαρμοστούν οι μέθοδοι που αναφέρουμε.

Στη συγκεκριμένη εργασία ο κώδικας είναι έτσι χτισμένος ώστε στη main να μπορούμε να φτιάχνουμε πολλά δένδρα ανάλογα με τις μεταβλητές που θα βάλουμε. Στο μέλλον θα μπορούσαμε να εξελίξουμε τον ήδη υπάρχοντα κώδικα και να αναπαραστήσουμε πολλούς πίνακες γειτνιάσεως σε μόνο ένα δένδρο. Όπου τα παιδιά της ρίζας θα αποτελούν έναν ολόκληρο πίνακα γειτνιάσεως. Επίσης, θα μπορούσαμε να χρησιμοποιήσουμε αυτές τις μεθόδους συμπίεσης που αναπτύξαμε σε διάφορες εφαρμογές, οι οποίες θα αποτελούνταν από πάρα πολλά δεδομένα και θα είχαμε την ανάγκη να δεσμεύουμε όσο γίνεται λιγότερο χώρο για την υλοποίησή τους. Θα ήταν χρήσιμο να εξελίξουμε τον κώδικα έτσι ώστε να δεσμεύουμε ακόμη λιγότερο χώρο για τον T_init (που είναι ο πίνακας που περιέχει όλα τα στοιχεία του δένδρου μας ανά επίπεδο), ώστε να χωρούν και να μπορούν να επεξεργαστούν σε έναν τυπικό υπολογιστή πολύ μεγάλοι πίνακες γειτνιάσεως. Τέλος, θα μπορούσαμε να εξελίξουμε τον κώδικα έτσι ώστε να μη λειτουργεί μόνο για μηδενικά ή άσσους αλλά και για άλλους αριθμούς. Και θα μπορούσε κάλλιστα κάθε κελί, που περιέχει διάφορους αριθμούς, μέσα στον αρχικό μας να αντιπροσωπεύει έναν ολόκληρο υποπίνακα με μηδενικά ή άσσους

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1]. N. R. Brisaboa, S. Ladra, and G. Navarro. K^2-trees for compact Web graph representation. In Proc. 16th Int. Symposium on String Processing and Information Retrieval (SPIRE 2009), LNCS vol. 5721, pp. 18–30, 2009.

[2]. De Bernardo G., Álvarez-García S., Brisaboa N.R., Navarro G., Pedreira O. Compact Querieable Representations of Raster Data, Proc. of the 20th Int. Symposium on String Processing and Information Retrieval (SPIRE 2013), Vol. 8214 -> LNCS vol. 8214, pp. 96-108, 2013.

[3].Wikipedia. https://en.wikipedia.org/wiki/Webgraph

[4]. Wikipedia. https://en.wikipedia.org/wiki/Web_page

[5]. Wikipedia. https://en.wikipedia.org/wiki/Hyperlink

[6]. Wikipedia. https://en.wikipedia.org/wiki/Server_(computing)

[7]. D. Donato, S.Millozzi, S.Leonardi, P.Tsaparas, Mining the inner structure of the Webgraph, in: Proceedings of 8thWorkshop on the Web and Databases (WebDB), pp. 145–150, 2005.

[8]. Wikipedia. https://en.wikipedia.org/wiki/Depth-first_search

[9]. Wikipedia. https://en.wikipedia.org/wiki/Breadth-first_search

[10]. J. Kleinberg, R.Kumar, P.Raghavan, S.Rajagopalan, A.Tomkins, The Web as a graph : measurements, models, and methods, in : Proceedings of 5th Annual International Conference on Computing and Combinatorics (COCOON), Lecture Notes in Computer Sciences, vol. 1627, pp.1–17, 1999 .

[11]. Webopedia. http://www.webopedia.com/TERM/W/webspam.html

[12]. L. Becchetti, C. Castillo, D. Donato, R. Baeza-Yates, S. Leonardi, Link analysis for Web spam detection, ACM Transactions on the Web 2, 2008.

[13]. P.Boldi, S.Vigna, The WebGraph framework I:compression techniques ,in : Proceedings of 13th Internationa lWorld Wide Web Conference (WWW) , pp. 595–601, 2004.

[14]. Y.Asano, Y.Miyawaki, T.Nishizeki, Efficient compression of Web graphs, in : Proceedings of 14th Annual International Conference on Computing and Combinatorics (COCOON),Lecture Notesin Computer Science, vol. 5092, pp. 1–11, 2008.

[15]. A. Apostolico, G.Drovandi, Graph compression by BFS, Algorithms 2 1031–1044, 2009.

[16]. Wikipedia. https://en.wikipedia.org/wiki/Breadth-first_search

[17]. G. Buehrer, K.Chellapilla, A scalable pattern mining approach to Web graph compression with communities, in : Proceedings of 1st ACM International Conference on Web Search and Data Mining (WSDM),pp.95–106, 2008.

[18]. Wiktionary https://en.wiktionary.org/wiki/biclique

[19]. Wikipedia. https://en.wikipedia.org/wiki/Code::Blocks

[20]. Wikipedia. https://en.wikipedia.org/wiki/C_(programming_language)

[21]. Wikipedia. https://en.wikipedia.org/wiki/Adjacency_matrix

[22]. Wikipedia. https://en.wikipedia.org/wiki/Tree_height_measurement

[23]. Wikipedia. https://en.wikipedia.org/wiki/Tree_(data_structure)

[24]. H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann Publishers Inc., 2006.

[25]. https://docs.oracle.com/cd/E19182-01/820-3742/ghpow/index.html

[26]. Wikipedia. https://en.wikipedia.org/wiki/Mask_(computing)

[27]. Stack over flow. https://stackoverflow.com/questions/1733881/c-correctly-freeing-memory-of-a-multi-dimensional-array

[28]. Exploring Binary. http://www.exploringbinary.com/ten-ways-to-check-if-an-integer-is-a-power-of-two-in-c/

[29]. Geeks for geeks. http://www.geeksforgeeks.org/program-to-find-whether-a-no-is-power-of-two/

[30]. Stack over flow. https://stackoverflow.com/questions/2525310/how-to-define-and-work-with-an-array-of-bits-in-c

[31]. Stack over flow. https://stackoverflow.com/questions/917783/how-do-i-work-with-dynamic-multi-dimensional-arrays-in-c

[32]. Stack over flow. https://stackoverflow.com/questions/111928/is-there-a-printf-converter-to-print-in-binary-format

[33]. Stack over flow. https://stackoverflow.com/questions/5248915/execution-time-of-c-program

[34]. Stack over flow. https://stackoverflow.com/questions/17042476/cmd-export-all-the-screen-content-to-a-text-file

[35]. Tutorials point. https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm

[36]. Tutorials point. http://www.tutorialspoint.com/online_c_formatter.htm

[37]. Cplusplus. http://www.cplusplus.com/forum/unices/50561/

[38]. http://www.cs.loyola.edu/~jglenn/702/S2008/Projects/P3/time.html

[39]. https://linux.die.net/man/2/gettimeofday

[40]. Tutorialspoint. https://www.tutorialspoint.com/cprogramming/c_pointers.htm

[41]. Tutorialspoint. https://www.tutorialspoint.com/cprogramming/c_structures.htm

[42]. c4learn. http://www.c4learn.com/data-structure/c-program-to-create-singly-linked-list/

[43]. Cprogramming.com. http://www.cprogramming.com/snippets/source-code/singly-linked-list-insert-remove-add-count

[44]. Tutorialspoint. https://www.tutorialspoint.com/c_standard_library/time_h.htm

[45]. Tutorialspoint. https://www.tutorialspoint.com/c_standard_library/math_h.htm

[46]. Tutorialspoint. https://www.tutorialspoint.com/c_standard_library/c_function_free.htm

[47]. https://gcc.gnu.org/onlinedocs/cpp/Macro-Arguments.html

[48]. Thegeekstuff. http://www.thegeekstuff.com/2013/04/c-macros-inline-functions/?utm_source=tuicool

[49]. Stackoverflow. https://stackoverflow.com/questions/8559946/nested-macro-order-of-expansion

[50]. Codecall. http://forum.codecall.net/topic/62497-min-and-max-in-a-rand/

[51].Cplusplus. http://www.cplusplus.com/reference/cstdlib/rand/

[52]. WebGraph. http://webgraph.di.unimi.it/